NTNU
Det skapende universitet

Eurobot 2007

Gunnar Kjemphol

Master i teknisk kybernetikk
Oppgaven levert: Juni 2007
Hovedveileder: Sverre Hendseth, ITK

Norges teknisk-naturvitenskapelige universitet
Institutt for teknisk kybernetikk

Oppgavetekst

Utvikling av autonom robot for deltagelse i Eurobot Open 2007

Oppgaven gitt: 08. januar 2007
Hovedveileder: Sverre Hendseth, ITK

Masteroppgave

Eurobot 2007

Av:

Gunnar Kjemphol

NTNU 2007

Sammendrag

Eurobot Open er en internasjonal robotkonkurranse for studenter og uavhengige robotklubber fra hele verden, der lagene konkurrerer mot hverandre med autonome roboter. Konkurransen finner hvert år sted i et europeisk land, og ble dette året avholdt i La Fertè-Bernard i Frankrike. NTNU har vært representert siden år 2000. Oppgaven for konkurransen er ny hvert år. Årets oppgave som har fått navnet «Robot Recycling Rally», går på resirkulering av flasker, bokser og batterier. Robotene skal i løpet av 90 sekunder samle og sortere flest mulig flasker, bokser og batterier.

Målet med masteroppgaven har vært ferdigstillelse av roboten som var påbegynt i prosjektet høsten 2006, slik at denne kunne delta i Eurobot Open 2007. Hovedfokus har gjennom arbeidet vært utviklingen av en modulbasert robot, hvor modulene kommuniserer ved hjelp av CAN-buss. CAN-buss systemet fra prosjektet er blitt videreutviklet fra prosjektperioden, ved blant annet hardware meldingsfiltrering. Noder implementert med meldingsfiltrering, mottar i software bare de meldinger som hardwarefilteret slipper igjennom.

Robotens motordrivermoduler er videreutviklet fra prosjektperioden. Modulene er nå mer robust enn tidligere, da disse har blitt oppgradert med flere støyfiltere. Modulene styres ved hjelp av CAN-buss, og kan regulere motorer ved hjelp av enten hastighetsregulering eller momentregulering. Hastigheten og momentet motorene utvikler kan også til en hver tid leses ut ved hjelp av CAN-bussen.

Et robot strategiplanleggingsystem implementert i C++ er utviklet for roboten. Strategiplanleggingsystemet er basert på «action selection»-prinsippet, som velger en av mange strategier basert på prioriteter. Rammeverket til systemet er dynamisk da nye strategier kan legges til eller fjernes uten at dette påvirker hverken rammeverk eller eksisterende strategier. Alle strategier i systemet blir automatisk oppdatert når planleggingsystemet mottar en ny observasjon, slik at alle disse har samme og siste oppdaterte informasjon om verden de opererer i.

Operativsystemet Ubuntu Linux har vært installert på robotens hoveddatamaskin. Hoveddatamaskinen består av et Commell LV-677 hovedkort , Intel Core 2 Duo prosessor på 2 Ghz, og 1 GB ram. I løpet av test- og utviklingsperioden oppsto en del uforutsette problemer med hovedkortet, men etter at disse ble rettet har løsningen fungert utmerket.

Chassis for roboten er bygget etter maksimal størrelse Eurobotreglene tillater. Denne størrelsen er lik hvert år, og det har vært gjort en stor innsats i å bygge et chassis som kan benyttes over flere år. Viktige kriterier har også vært enkel montering og demontering, av moduler inne i roboten.

Undertegnede har fungert som prosjektleder, med mye samarbeid med de andre prosjektdeltagerne, og diverse administrative oppgaver.

Finansiering av prosjektet er skaffet gjennom sponsoravtale med Kongsberg Gruppen ASA, og midler fra Institutt for Teknisk Kybernetikk. Kongsberg Gruppen ASA ønsket å være eneste sponsor, og har dermed sponset hele prosjektet.

Roboten gjorde det ikke så bra som ønsket i konkurransen. Medvirkende årsaker til dette var blant annet uforutsette hendelser med robotens grunnleggende systemer i de siste ukene før avreise. Disse hendelsene medførte mye ekstraarbeid, og gjorde at det ikke ble nok tid igjen til skikkelig testing av hele systemet samlet.

Undertegnede er stolt av arbeidet som er lagt ned i prosjektet, både måten prosjektet er gjennomført på og hvordan roboten er bygget. Det er hele tiden vært viktig med gjenbrukbare moduler, alt fra programmer og hardware, til mekanisk konstruksjon. Fortsetter Eurobotlagene å utvikle moduler på denne måten, kan Institutt for Teknisk Kybernetikk bygge opp et lager av moduler som kan brukes år etter år. På denne måten kan NTNU i nærmeste fremtid hevde seg i Eurobot konkurransen.

Forord

Denne rapporten er skrevet av Gunnar Kjemphol våren 2007.

Prosjektet er finansiert ved hjelp av sponsormidler fra Kongsberg Gruppen ASA, gjenværende midler fra tidligere år, og støtte fra Instituttet for Teknisk Kybernetikk.

Undertegnede ønsker å takke Kristian M. Knausgård og Lars Vråle, for samarbeidet med utviklingen og konstruksjonen av roboten. Videre ønsker jeg å takke Eksperter i Team gruppen for utviklingen av sorteringsmodulen til roboten. Sverre Hendseth fortjener en stor takk for veiledning gjennom hele prosjektet og masteroppgaven. Hans Jørgen Berntsen og Terje Haugen har vært behjelpelig med mekanisk utvikling av roboten, jeg takker også dem.

Gunnar Kjemphol

Trondheim 19. juli 2007

Innhold

Innhold **vii**

1 Innledning **1**

2 Bakgrunn **3**
2.1 Regler 2007 . . . 3
2.2 Eurobot ved Institutt for Teknisk Kybernetikk . . . 4

3 Teori **7**
3.1 Planleggingssystemer og Kunstig Intelligens . . . 7
3.1.1 Hierarkiske paradigmet . . . 7
3.1.2 Reaktive paradigmet . . . 8
3.1.3 Ruteplanlegging . . . 10
3.2 Spenningsforsyning . . . 11

4 Hoveddel Robot **13**
4.1 Introduksjon av roboten . . . 13
4.2 Grunnleggende Systemer på Roboten . . . 14
4.2.1 Innebygget Hoveddatamaskin . . . 14
4.2.2 Motor pådragssystem . . . 15
4.2.3 Intern Spenningsforsyning . . . 17
4.2.4 Antikollisjon . . . 18
4.2.5 IO Kort . . . 19
4.2.6 Flaske-/Boks-sorterer . . . 21
4.2.7 Batteri sorterer . . . 22
4.2.8 Servoer . . . 24
4.2.9 Batterier . . . 24
4.2.10 Kontakter . . . 25
4.3 Programvare . . . 26
4.3.1 Operativsystemet . . . 26
4.3.2 CAN-buss . . . 26
4.3.3 Posix-kø systemet . . . 29
4.3.4 Strategiplanleggingssystem . . . 30
4.3.5 Navigasjonsystem . . . 38

4.3.6 Bildebehandlingsystem . . . 40
4.3.7 Proxy- / Gateway-system . . . 41
4.4 Mekanisk konstruksjon . . . 43
4.4.1 Chassis . . . 43
4.4.2 Hoveddatamaskin . . . 43
4.4.3 Hjul og Motor-/gir-moduler . . . 44
4.4.4 Antikollisjonssensorer . . . 46
4.4.5 Kamera . . . 46
4.4.6 Koblingsmodul . . . 46

5 Hoveddel Prosjektledelse **49**
5.1 Organisering . . . 50
5.1.1 Reise . . . 50
5.1.2 Sponsor . . . 50
5.1.3 Budsjett og konto . . . 51
5.1.4 Eurobot-forenings medlemskap . . . 51
5.2 Utførsel av Robot og utstyr til EU land . . . 51
5.3 Utførsel av varer til reperasjon . . . 52

6 Resultater og Diskusjon **53**
6.1 Eurobot Open 2007 . . . 53
6.1.1 Konkurransen . . . 53
6.1.2 Diverse . . . 56
6.2 Grunnleggende Systemer på Roboten . . . 57
6.2.1 Innebygget Hoveddatamaskin . . . 57
6.2.2 Motor Pådragssystem . . . 57
6.2.3 Intern Spenningsforsyning . . . 58
6.2.4 Antikolisjon . . . 58
6.2.5 IO Kort . . . 59
6.2.6 Flaske-/Boks-sorterer . . . 59
6.2.7 Batteri Sorterer . . . 59
6.2.8 Servoer . . . 60
6.2.9 Batterier . . . 60
6.2.10 Kontakter . . . 60
6.3 Programvare . . . 61
6.3.1 Operativsystemet . . . 61
6.3.2 CAN-Buss . . . 61
6.3.3 Posix-kø-systemet . . . 62
6.3.4 Strategiplanleggingssystemet . . . 62
6.3.5 Navigasjonssystemet . . . 66
6.3.6 Bildebehandlingssystemet . . . 66
6.3.7 Proxy- /Gateway-systemet . . . 66
6.4 Mekanisk konstruksjon . . . 68
6.4.1 Chassis . . . 68

6.4.2 Motor-/Gir-moduler . . . 68
6.4.3 Antikollisjonssensorer . . . 69
6.4.4 Kamera . . . 70
6.4.5 Koblingsmodul . . . 70

7 Resultater og Diskusjon Prosjektledelse 71

8 Diverse 73
8.1 Kretskortdesign . . . 73
8.2 Anbefalinger til NTNU for Eurobot videre . . . 73
8.3 Spillebordet . . . 75
8.4 Omvisninger . . . 75
8.5 Litteraturstudie og læringsandel . . . 76

9 Konklusjon 77

Bibliografi 79

A BIOS-oppgradering 83

B Installere Ubuntu på COMMELL LV-677 FlashDisk 85
B.1 Formatering av CompactFlash-kortet . . . 85
B.2 BIOS-oppsett . . . 85
B.3 Under installasjon . . . 86
B.4 Autologin . . . 86

C Installere PCAN-driver på Ubuntu-server 87

D CAN-BUSS-Adresser 89

E Møtereferat fra møte med Keith Downing 91

F Produksjon av printkort 93

G Proforma Faktura 95

H Grensesnitt 97

I Kretstegninger 101

J CD-ROM 113

Kapittel 1

Innledning

Eurobot Open er en internasjonal robotkonkurranse, som hvert år arrangeres i et europeisk land. Lag fra forskjellige land konkurrerer mot hverandre med autonome roboter. Hvert år forandres reglene så ikke nøyaktig den samme roboten kan benyttes år etter år. Årets oppgave har fått navnet «Robot Recycling Rally», og går ut på sortering av flasker bokser og batterier. NTNU og Teknisk Kybernetikk har vært representert i Eurobot hvert år siden år 2000.

Denne rapporten dokumenterer arbeidet som er gjort både mekanisk, software og hardware, for å ferdigstille roboten for deltagelse i Eurobot Open 2007.

I løpet av denne masteroppgaven skal det utvikles en autonom robot for deltagelse i konkurransen Eurobot Open 2007. Det skal fokuseres på ferdigstillelse av roboten. Et planleggingsystem som skal kunne brukes både på årets og senere års roboter skal utvikles. Planleggingssystemet skal kunne benytte flere uavhengige strategier. Motordrivermodulen utviklet under prosjektperioden høsten 2006, skal videreutvikles og benyttes på roboten. En modulbasert robot skal konstrueres ved hjelp av posix-køer mellom programmene på hoveddatamaskinen og CAN-buss mellom de forskjellige modulene.

Undertegnede har vært prosjektleder for robotutviklingen, og har av denne grunn valgt å omtale hele roboten i denne rapporten. Moduler ikke utviklet av undertegnede vil presenteres i rapporten med funksjonalitet og grensesnitt. Disse modulene er viktig å få med da spesielt planleggingsystemet benytter disse. Som prosjektleder og ansvarlig for ferdigstillelse av roboten, har også undertegnede vært avhengig av å vite hvordan alle moduler fungerer.

Rapporten gir først en innføring i Eurobot konkurransen, før reglementet for 2007 blir presentert. Resultater fra de åtte årene NTNU har deltatt i konkurransen blir også listet opp her.

Kapittel tre gir teori på planleggingssystemer, og «kunstig intelligens». Her er det det hierarkiske og det reaktive paradigmet innen «kunstig intelligens» som er viet fokus. Deretter gis teori på måter for ruteplanlegging. Teori på spenningsforsyninger er også tatt med.

Hoveddelen av rapporten er delt i to, da den første omhandler konstruksjon

av roboten, og den neste prosjektledelse.

Hoveddelen av rapporten som går på robotkonstruksjonen, er delt opp i tre deler, etter at en introduksjon av roboten er gitt, Grunnleggende systemer, Programvare, og Mekanisk konstruksjon.

- Grunnleggende systemer, beskriver de fleste modulene som inneholder logikk på roboten.
- Programvare, presenterer programmene som er utviklet for roboten.
- Mekanisk konstruksjon, forklarer mekanisk oppbygging av roboten.

En hoveddel som presenterer prosjektledelse er også tatt med. Resultater og diskusjon er viet et kapittel for hver av hoveddelene.

Et diverse kapittel er tatt med til oppgaver som ikke faller naturlig inn under prosjektets hovedfokus.

Konklusjonen for prosjektet er gitt i kapittel ni.

Kapittel 2

Bakgrunn

Eurobot Open er en internasjonal robotkonkurranse som arrangeres hvert år i et europeisk land. Arrangementet feirer i år 10-års jubileum og arrangeres dette året i La Fertè-Bernard, der det hele startet i 1998. Konkurransen er åpen for studenter og uavhengige robotklubber fra hele verden. Deltagernes aldersgrense er 30 år, med dispensasjon for en eldre deltager.

Reglene for konkurransen er nye hvert år, slik at samme robot ikke kan delta flere år på rad uten ombygging. Hovedtrekkene er derimot de samme hvert år, der to autonome roboter konkurrerer mot hverandre på et spillebord. Konkurransen arrangeres som en cup der man starter med innledende runder som fører frem til finaler. For å delta i cupen må roboten gjennom en kvalifiseringsrunde. Kvalifiseringsrunden går på fysisk størrelse og funksjoner, samt at den må kunne vinne et spill uten motstand.

2.1 Regler 2007

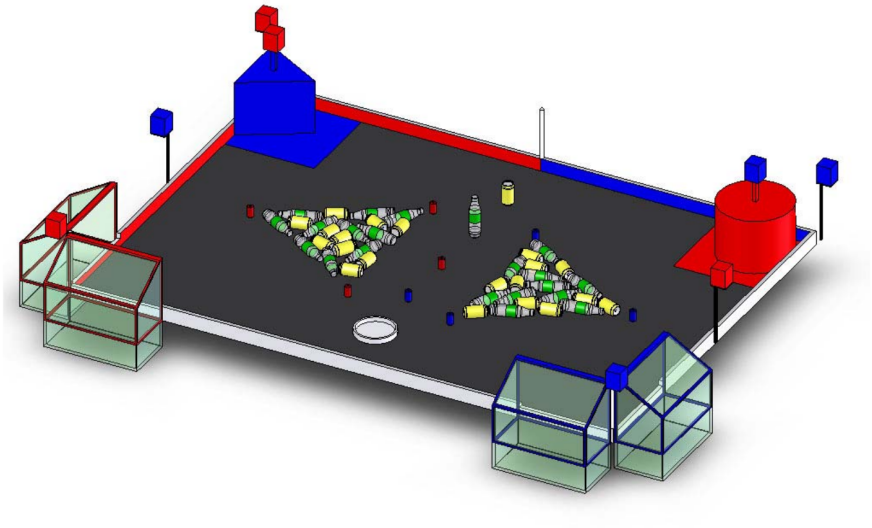
Reglementet for konkurransen 2007 går på sortering av flasker, bokser og batterier. Konkurransen er ut fra oppgaven gitt navnet "Robot Recycling Rally". Spillebordet er 300 centimeter langt og 210 centimeter bredt. På utsiden av spillebordet er det festet fire beholdere for sortering av flasker og bokser, to beholdere for hvert lag. Flasker og bokser skal sorteres i hver sine beholdere. Beholderene som tilhører et lag er lokalisert på motsatt side av bordet hvorfra roboten starter. Batterier finnes i to forskjellige farger, en farge for hvert lag, og skal sorteres i en felles beholder som kan ha en av tre posisjoner på spillebordet.

Flasker, bokser blir før start av et spill plassert i to triangler på spillebordet. Det finnes 10 flasker og 10 bokser i hvert triangel, i tillegg til en flaske og en boks som står plassert på midtlinjen av spillebordet. Det er fire batterier for hvert lag, hvor to av disse er plassert på midtlinjen og de resterende i trianglenes hjørner. Oppsettet av spillebordet er vist i figur 2.1.

Et spill varer i 90 sekunder, der roboten skal utføre alle handlinger auto-

nomt. En 50 centimeter lang startsnor brukes for å starte roboten, og når tiden for et spill er ute skal alle aktuatorer stoppe.

Det stilles strenge krav til roboten både når det gjelder omkrets og høyde, samt hva som er et minimum av implementert sikkerhetutstyr. Roboten kan ikke ha omkrets større enn 120 centimeter før start av et spill. Så snart spillet er i gang kan roboten folde seg ut til en omkrets på maksimalt 140 centimeter. Høyden av roboten kan ikke overstige 35 centimeter, da ikke medregnet obligatorisk tårn og nødstoppbryter. Tårnet skal være 80x80 mm og ha en høyde fra bakken på 43 centimeter. Nødstoppbryteren skal stoppe alle aktuatorer. Antikollisjonsystem er også påkrevd, og dette testes under kvalifiseringsrunden.



Figur 2.1: Oppsett spillebord 2007

2.2 Eurobot ved Institutt for Teknisk Kybernetikk

Institutt for Teknisk Kybernetikk (ITK) har deltatt i Eurobot Open siden år 2000. Lagene har som oftest vært sammensatt av diplomstudenter. Noen av årene har også en Eksperter i Team gruppe vært involvert.
Plasseringene som ITK har oppnådd i Eurobot Open er som følger:
(Garsjo, Sondre og Platou, Halvor, 2006)

Årstall og Konkurransenavn	**Plassering**	**Antall finalelag (antall lag totalt)**	**Prosentvis plassering**
2000 Fun Fair	8.	13	62 %
2001 Space Odyssey	13.	18	72 %
2002 Flying Billiards	17.	25	68 %
2003 Heads or Tails	16.	32	50 %
2004 Coconut Rugby	21.	41	51 %
2005 Bowling	11.	50	22 %
2006 Funny Golf	44.	50	88 %
2007 Robot Recycling Rally	25.	39	64 %

Kapittel 3

Teori

3.1 Planleggingssystemer og Kunstig Intelligens

Kunstig intelligens og planleggingssystemer i roboter deles ofte inn i to paradigmer, hierarkiske, og reaktive. De to paradigmene vil her presenteres.

3.1.1 Hierarkiske paradigmet

Den første måten å tilnærme seg kunstig intelligens i roboter er den hierarkiske (Murphy, 2000). Det som kjennetegner det hierarkiske paradigmet er måten en roboten planlegger på. Det første roboten gjør er å benytte sensorinformasjon, som for eksempel kan være et kamera. Ved hjelp av sensorinformasjonen planlegges handlinger, før roboten deretter utfører ønsket handling. Når handlingen er utført gjentas sekvensen på nytt. (SENSE -> PLAN -> ACT).

STRIPS

STRIPS står for "Stanford Research Institute Problem Solver", og er et planleggingsystem under det Hierarkiske paradigmet.

Strips er en algoritme som kan benyttes til å løse generelle problemer knyttet til kunstig intelligens. Algoritmen prøver å løse et globalt problem. Klarer den ikke å finne en løsning på dette målet på en iterasjon, løser den et delmål. Etter at dette delmålet er løst, vil det globale målet ha redusert kompleksitet og en enklere løsning. Måten Strips tilnærmer seg en global løsning på er inspirert av menneskets kognitive tankegang, der delmål løses for å komme frem til ønsket resultat (SRI, 1970) (Murphy, 2000).

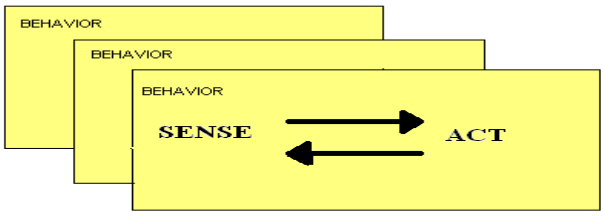
STRIPS ble benyttet til problemløsning på blant annet roboten Shakey (SRI, 2007b), som ble utviklet av SRI International (SRI, 2007a) fra 1966 til 1972. Shakey var den første roboten som kunne resonere over egne handlinger.

Kognitiv aktivitet

Kognitiv aktivitet inkluderer både tilbakekobling og foroverkoblingskontroll, der en robotagent finner en feil i hva den ønsker å gjøre og hva som faktisk skjedde (Murphy, 2000). En kognitiv aktivitet kan være å bestemme hva en robot skal lete etter eller gjøre neste gang den er klar til ny handling.

3.1.2 Reaktive paradigmet

Rodney Brooks, professor på MIT, la grunnlaget for det reaktive paradigmet. Paradigmet er inspirert av biologisk intelligens, som baserer seg på å føle for så å handle, i flere lag (SENSE -> ACT), se figur 3.1. Her finnes ikke planleggingsdelen (PLAN) som det hierarkiske paradigmet har. Paradigmet foreslår en vertikal oppdeling med flere lag liggende vertikalt, der hvert lag har sin oppførsel/tilstandsmaskin. Utførelsen av hvert lag blir utført i parallell eller i serie. Ett lag kan også ha høyere prioritet enn andre lag og undertrykke disse (Murphy, 2000).



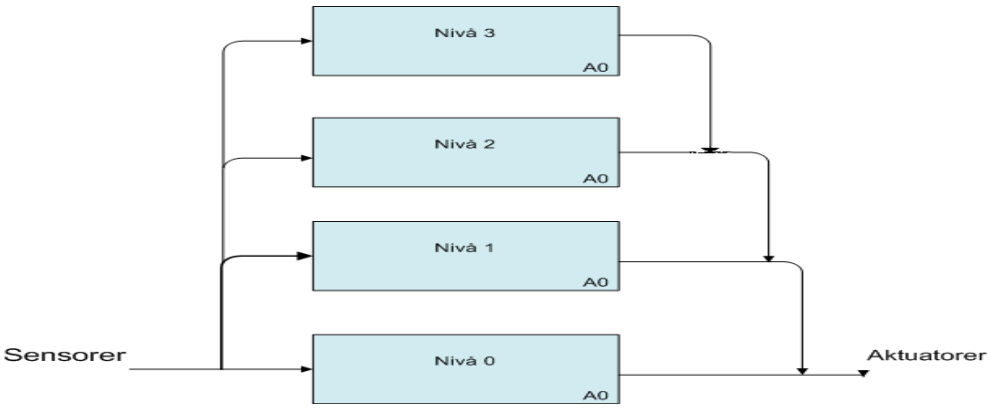
Figur 3.1: Reaktive paradigmet

Det reaktive paradigmet brukes i hovedsak der all informasjonen roboten har tilgjengelig om området den opererer i, ikke til en hver tid er oppdatert, og når utfallet av en handling ikke er sikker (Liao C., Yo R., Chen Z., Chan D., Fu L., 2005). Det er i hovedsak to arkitekturer som benyttes for reaktive roboter. Dette er «Subsumption Architecture» og «Selection Algorithms».

Subsumption Architecture

«Subsumption»-arkitekturen er en lagdelt arkitektur. Hvert lag består av en tilstandsmaskin som ligger mellom sensorer og aktuatorer. Arkitekturen er vertikalt oppdelt, og består av horisontale lag, der det laveste nivå består av den mest primitive oppførsel. Lagene får mer og mer avansert oppførsel etter lenger opp man kommer (Rodney A. Brooks, 1985). Utgangene av hvert lag er

enkle funksjoner av inngangssignalet og lokale variabler. Høyere lag inneholder automatisk lavere lag i modellen, som fører til at lave nivå fungerer fint uten høyere nivåer, se figur 3.2



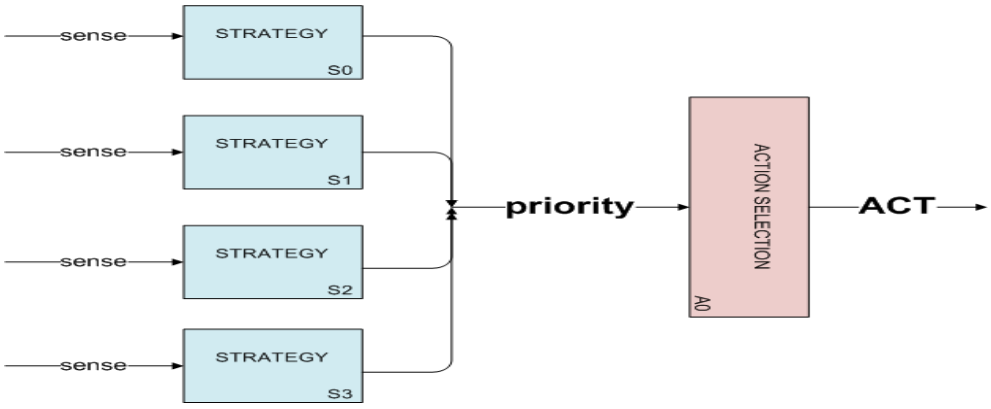
Figur 3.2: Subsumption-arkitekturen

Allen var den første roboten som benyttet seg av subsumption-arkitekturen, og ble bygget av blant annet Rodney Brooks.

Action Selection

«Action Selection» baserer seg på en algoritme, som til en hver tid velger ut en strategi eller handling fra en av flere moduler. Modulene kan være implementert på flere forskjellige måter, blant annet som tilstandsmaskiner, eller selvlærende moduler. I «Action Selection»-arkitekturen kan man ha mange strategier i hver sin modul, som velges basert på strategienes momentane prioritet. Modulen gir neste handling som skal utføres (Stan Franklin, 1995).

I «Action Selection»-arkitekturen, er ikke de forskjellige strategiene, og lagene oppdelt som i «Subsumption»-arkitekturen, hvor de ligger i horisontale lag som får mer og mer avansert oppførsel etter lenger opp man kommer. I «Action Selection»-arkitekturen kan man ha mange strategier hvor hver strategi har sin egen kompetanse, se figur 3.3. Strategiene velges basert på strategienes momentane prioritet. Et viktig kriterium for strategiene i «Action Selection»-arkitekturen er at disse til en hvert tid får oppdatert informasjon om verden de opererer i, for å kunne gi riktig prioritet. «Action Selection»-arkitekturen er attraktiv på grunn av modularitet, fleksibilitet, og robusthet. Strategier kan legges til og fjernes uten at dette påvirker de andre strategiene. (Pattie Maes, 1989)



Figur 3.3: Action Selection-arkitekturen

3.1.3 Ruteplanlegging

Metrisk ruteplanlegging

Metrisk ruteplanlegging konverterer den fysiske verden, som da gjerne består av roboten og diverse hindringer over i et konfigurasjonsrom. Den fysiske verden består av en gitt mengde med frihetsgrader, som beskriver blant annet hvor et objekt befinner seg, retning og helling på dette. Hvor objektet er plassert er gjerne gitt i kartesiske koordinater (x, y, z). Hvilken retning det har og om objektet har en hellingsgrad er ofte gitt i Eulervinkler, pitch, yaw, og roll (ϕ, θ, γ). En god konfigurasjonsromrepresentasjon har redusert verdens frihetsgrader ned til gjerne to, og ikke mer enn tre. For en robot som skal bevege seg på bakkenivå, og man ikke trenger å vite retningen på roboten eller en hindring, får man en konfigurasjonsromrepresentasjon på bare to dimensjoner. Man trenger ikke vite hvilken retning en hindring har om man skal planlegge en rute som går rundt denne. En metrisk ruteplanlegger for roboter forutsetter ofte et konfigurasjonsrom der man representerer roboten og hindringer med bare to frihetsgrader. For at en robot skal kunne ha to frihetsgrader må den være holonom. Holonom vil si at roboten har like mange eller flere frihetsgrader enn verden den skal operere i. Roboter som har to frihetsgrader moduleres gjerne som runde, og de må kunne snu om sin egen akse (Murphy, 2000). Har roboten flere frihetsgrader enn verden den skal operere i, vil den også beskrives som redundant.

Grid verdens representasjon

Grid representasjon av verdenskordinatene er en av de vanligste måtene å representere kjørbart område på. En forutsetning for å kunne benytte denne verdensrepresentasjonen, er at planleggingssystemet har tilgjengelig et oppdatert kart av området.

A-Star

A-Star er en av de mest kjente metriske ruteplanleggere, som finner en rute fra en startnode til en målnode.

A-Star algoritmen er effektiv på å forkaste ikke optimale ruter raskere enn de fleste andre algoritmer som brukes i metrisk ruteplanlegging. Algoritmen benytter seg av et heuristikk estimat, som estimerer hvor langt det er til målnoden. Dette estimatet tas hver gang en ny node er nådd, slik at estimatet blir kortere for hver gang den nærmer seg målnoden. A-Star algoritmen finner ikke bare en rute om der er en, men den finner også den korteste.

Er eneste rute til målnoden en som først går bort fra målnoden, vil algoritmen bruke lang tid på å finne beste rute. Algoritmen prøver da først alle ruter som kan virke som om de har kortest vei, da disse er kortest i luftlinje (Heyes-Jones, 2007b) (Wikipedia, 2007).

3.2 Spenningsforsyning

Likespenning-spenningsforsyninger for elektronisk utstyr finnes i hovedsak i to forskjellige varianter, lineære- og svitsjede-spenningsregulatorer.

De lineære spenningsregulatorene har lineært tap proporsjonalt med spenning og strøm. Effekttapet kan regnes ut etter følgende formel:

$$P_{tap} = U_{spenningsreg}I = (U_{inn} - U_{last})I \tag{3.1}$$

Effekttapet i en lineær spenningsregulator blir overført til varme, og disse trenger kjøling.

Svitsjede-spenningsregulatorer har et betydelig mindre effekttap enn lineære. En svitsjet-spenningsregulator har en intern transistorkrets. Denne er som oftest en MOSFET transistor som skrur av og på spenningen med en høy frekvens, rundt 100khz. På denne måten flyttes energi fra inngangen av kretsen til utgangen i pulser. Ved å benytte denne måten til spenningsomforming, vil man få en effektiv spenningsforsyning med en effektivitet på fra omtrent 80 til 90 prosent. Ved å benytte effektoverføring fra inngang til utgang som en svitsjet spenningsregulator, er kravet til inngangspenning ikke så fastsatt som ved lineære regulatorer. Ved hjelp av en svitsjet spenningsforyning kan man blant annet levere høyere spenning ut av regulatoren enn inngangsspenningen. Inngangspenningen kan også være mange ganger så høy som utgangspenningen, uten at dette fører til større effekttap. (Semiconductor, 2002)

Kapittel 4

Hoveddel Robot

4.1 Introduksjon av roboten

Under hele utviklingsfasen, fra høsten 2006 og frem til midten av mai 2007, er det lagt ned mye arbeid i å designe en robot som er mest mulig modulbasert, dette gjelder både hardware og programvare. Ved å designe roboten modulbasert vil man på en grei måte kunne bytte ut enheter som ikke fungerer tilfredsstillende, uten at dette påvirker resten av systemet. For kommunikasjon mellom de forskjellige hardwareenhetene er det benyttet CAN-buss kommunikasjon. Roboten har implementert sju CAN-bussnoder, som hver kontrollerer sin modul. På hoveddatamaskinen kjøres fem programmer for kontroll av roboten. De fem modulene er strategiplanleggingssystemet, navigasjonssystemet, to bildebehandlingsystem samt et proxy- / gateway-system, som tar seg av meldingssendingen både internt og eksternt fra hovedkortet.

Hele roboten er bygget med tanke på å kunne gjenbruke deler av denne ved senere års konkurranser. Dette gjelder alt fra hardware og software samt den mekaniske konstruksjonen av roboten. Dette anses som viktig for at NTNU skal kunne hevde seg i Eurobot Open i nærmeste fremtid. Ser man på de lagene som har vunnet konkurransen tidligere år, er det mye gjenbruk av systemer som har virket tidligere, og det er flere av de samme lagene som hevder seg år etter år.

Koordinatsystem

Det er to koordinatsystem roboten må ta hensyn til, disse reffereres til i literaturen som NED og BODY (Fossen, Thor I., 2002). Det ene er koordinatsystemet for spillebordet gitt i NED, der punktet (0,0) er i det blå hjørnet. Koordinaten x blir da langs langsiden av bordet, og y blir kortsiden. Det andre koordinatsystemet er sett i forhold til roboten BODY, der (0,0) er på midten av drivhjulakslingen. Koordinaten x er her fremover fra roboten, og positiv y er til høyre fra midtpunktet.

4.2 Grunnleggende Systemer på Roboten

4.2.1 Innebygget Hoveddatamaskin

Hoveddatamaskinen Seco PC/104 som har vært benyttet på tidligere års roboter ble valgt forkastet i løpet av prosjektperioden høsten 2006 (Kjemphol, Gunnar og Knausgaard, Kristian M, 2006). Et mini ITX hovedkort Commell LV-677 ble anskaffet sammen med en Intel Core 2 Duo 2Ghz prosessor og 1 GB ram. Den nye hoveddatamaskinen Commell LV-677 har det vært brukt mye tid på. Prosessoren som var valgt Intel Dual Core 2 var ikke støttet av BIOS versjon 1.31, som kortet ble levert med, til tross for at produsentens datablad bekreftet at den var støttet. Oppgradering av BIOS var ikke mulig uten først å bytte til en støttet prosessortype, noe som ikke var aktuelt, da denne i tilfelle måtte kjøpes. Teknisk avdeling hos Commell ble kontaktet, og ny BIOS versjon 1.32 ble bestilt direkte hos dem.

Ubuntu ble installert på hoveddatamaskinen og maskinen fungerte fint et par uker. Deretter oppsto det problemer under oppstartsprosedyren ved at maskinen låste seg, og operativsystemet ble ikke startet. Det var derimot mulig å komme inn i BIOS under oppstartsekvensen, samt foreta de forandringer man ønsket der. Problemene oppsto når man skulle lagre BIOS innstillingene i c-mos, da maskinen låste seg og de foretatte forandringer ble ikke lagret. Det viste seg at for hver gang man skulle starte maskinen på nytt, måtte c-mos kretsen nullstilles ved å kortslutte en jumper, for at maskinen skulle starte. Commell ble igjen kontaktet. De hadde uken før sluppet en ny BIOS versjon 1.4, og oppgradering til denne ble foretatt. Oppgradering av BIOS løste problemene, og tilsvarende feil har ikke oppstått siden. Prosedyre for oppgradering av BIOS er vedlagt i vedlegg B.2.

Tre uker før avreise til Frankrike, under testing av roboten, oppsto det igjen problemer med hoveddatamaskinen. Det var ikke mulig å koble seg til datamaskinen ved hjelp av ssh. Skjerm og tastatur ble koblet til og maskinen så ut til å virke fint. Nettverkskortet på datamaskinen var derimot ikke mulig å få kontakt med. Ethernet kontrolleren er en Intel 82573L Gigabit kontroller (Intel, 2007a). Intel har tilgjengelig diverse diagnoseverktøy for nettverks kontrollere som ble benyttet (Intel, 2007b). Ingen av programmene kunne rette feilen, da de bare klarte å detektere Ethernettkontrolleren, men ikke lese data fra denne. Nok en gang ble Commell kontaktet. Commell hadde ingen løsning på problemet, og det ble anbefalt å kontakte leverandøren av kortet for garantireperasjon.

Samtidig som ny BIOS ble bestilt, ble det også bestilt et ekstra hovedkort til roboten, for å ha dette om problemer med det som var installert skulle oppstå. Det ble besluttet å bytte hovedkortet. Prosessor og minne ble nå montert over på det nye kortet, og et nytt Compact Flash kort på 2GB ble satt inn. Deretter måtte operativsystem, alle drivere og programmene installeres på nytt. Det nye Compact Flash kortet ble satt inn for å kunne beholde programmene og

driverene som var installert, for det defekte kortet.

Hovedkortet kan kobles direkte til batterier, da dette har innebygd spenningsregulator. Spenningsregulatoren tolererer fra 8 til 21 volt inngangsspenning.

4.2.2 Motor pådragssystem

Motorpådragssytemet ble utviklet av undertegnede, som en del av prosjektoppgaven høstsemesteret 2006 (Kjemphol, Gunnar og Knausgaard, Kristian M, 2006). Under prosjektperioden var der bare laget en motordriver, og roboten hadde behov for to. Motordriveren var testet og funnet velegnet for roboten, så det ble besluttet å bruke denne typen motordriver videre. Det ble nå gjort små forandringer på motordriveren, da det blant annet var ønske om å bytte CAN-kontaktene D-Sub9 fra hunkontakter til hankontakter. Motordriveren fikk nå versjonsnummer 2.3, og er vist i vedlegg I.5 og I.6.

Under testfasen av roboten, ble motordriver versjon 2.2 benyttet på høyre side, og versjon 2.3 på venstre. Etter en del bruk, oppsto det problemer med motordriveren versjon 2.2. Feilsymptomene som ble observert var at roboten ble stående med hylelyder fra høyre motor etter kort kjøring. CAN-bussen gikk i feilmodus, og man fikk ikke sendt meldinger på bussen.

Feilsøking på systemet ble foretatt for å finne problemet. Da CAN-bussen gikk i feilmodus, var det ikke godt å vite hvor i systemet feilen lå. Systematisk feilsøking ble foretatt for å lokalisere feilen. Motordriverene ble koblet fra og testet hver for seg. Ved første test virket begge som om de var i orden da roboten ble kjørt fremover. Navigasjonssystemet kunne også kjøre roboten til gitte waypoints, før feilen oppsto.

Det viste seg etter å ha koblet fra motordriverene igjen, at høyre motordriver kortsluttet spenningsforsyningen for fem volt når motoren ble satt til å kjøre bakover. Dette førte igjen til omstart av alle CAN-Bussnoder. Roboten kunne altså kjøre fremover til gitte waypoint helt til den skulle regulere vinkelen ved å kjøre høyre motor bakover. Da oppsto problemet.

Feilsøking på motordriveren ble foretatt, og det be sett på hvorfor CAN-bussen gikk i feilmodus. Det ble ikke funnet noe galt med bussen, CAN-buss kontrolleren, eller mikrokontrolleren. H-Bro kretsen LMD18200 (National Semiconductor, 2005) som er implementert på motordriveren ble byttet. Deretter fungerte motordriveren igjen som ønsket. Det viste seg dermed at kortslutning i motordriverens H-Bro kunne føre til at hele CAN-buss systemet og fem volt spenningsforsyningen fikk problermer. H-Broen er ikke direkte knyttet til femvolt-spenningsforsyningen annet enn gjennom mikrokontrolleren. H-broen har også egne batterier til drift av motorene.

Motordriver versjon 3.0

En motordriver versjon 3.0 ble designet for roboten, etter feilen som hadde oppstått på versjon 2.2. Versjon 2.3 var stort sett samme motordriver som versjon 2.2, men med annen CAN-kontakt. Det ble nå lagt vekt på å gjøre motordriveren robust mot spennings- og støypulser. Den største oppgraderingen som ble gjort var å legge inn RC-filter mellom mikrokontrolleren og H-bro kretsen. Kretskort utlegget fikk også en del små forandringer. RC-filtrene består av en motstand på 100 ohm og en kondensator på 10nF, som raskt filtrerer spenningspulser fra motordriveren. Filtrering av spenningspulser fra motoren ble det nå også tatt høyde for, og en kondensator med størrelse 10nF ble koblet i parallell med motorklemmene, så nært motoren som mulig. Kretsdesignet for motordriver versjon 3.0 er vist i vedlegg I.7 og I.8.

Batteriene som er benyttet på roboten er av typen Lithium ion polymer, som er batterier med meget lav indre motstand, og kan uten problemer levere strøm opp mot 40 ampere. Da motordriverene ikke tåler mer strøm enn tre ampere, ble det besluttet å legge inn en effektmotstand på 0.24 ohm i serie med spenningstilføresen til H-broene, for å hindre motorene og H-broen å trekke utilsiktede høye strømpulser fra batteriene. Det benyttes vanligvis bly-batterier eller andre strømforsyninger med betydelig høyere indre motstand enn Lithium ion polymer batterier til slike H-Broer og motorer.På blybatterier ønsker man så liten indre motstand som mulig, men dette er uansett betydelig høyere motstand enn hos Lithium ion polymer batteriene. Motstanden vil nå simulere en høyere indre motstand i batteriet og filtrere eventuelle høye strømpulser sammen med kondensatoren over motoren.

Grensesnitt Motordriver versjon 3.0

Her vil grensesnittet for motordriveren sett fra posix køene og CAN-bussen presenteres. Ikke alle meldinger er implementert på posix-systemet. Gateway-systemet omgjør CAN-buss-meldingene til Posix-kø-meldinger som roboten benytter, og omvendt. Meldingsidentifikatorene vises med posix-nummeret først, deretter CAN-identifikatoren, hver enkelt melding er presentert med det definerte navnet av verdien fra filene *posixconfig.h* og *proxy_message_config.h* Motordrivermodulene har for meldinger de skal motta fått CAN adressene fra 0x110 til 0x120.

`MINOR_MOTOR_CONTROL_MESSAGE og CAN_ADDR_MOTOR_MAIN`
Denne meldingen sender strukten *motor_control_message* i posix-køsystemet. Meldingen inneholder hastighets verdien som ønskes for de to motorene. Fortegnet på de to hastighetsverdiene i strukten bestemmer retningen på motoren. CAN-meldingen som sendes inneholder fire byte, de to første bytene er for venstre motor, og de to neste for høyre. Første byte for en motor gir rettningen, der 0xFF er fremmover og 0x00 er bakover. Andre byte gir hastighetsverdien fra 0x00 til 0xFF.

`CAN_ADDR_MOTOR_RIGHT_READ_QUAD og CAN_ADDR_MOTOR_LEFT_READ_QUAD`
Kommandomelding som sendes til motordriveren. Denne returnerer en melding med data som inneholder kvadraturteller verdien siden sist samme melding ble sendt.

`CAN_ADDR_MOTOR_DRIVER_INIT`
Kommandomelding som initialiserer motordriveren.

`CAN_ADDR_MOTOR_EMERGENCY_STOP`
Etter mottak av denne meldingen må motordriveren motta meldingen `CAN_ADDR_MOTOR_DRIVER_INIT` for å kunne reagere på nye pådragsmeldinger.

`CAN_ADDR_MOTOR_INIT_QUAD`
Meldingen kjører initialiserings rutinen på kvadraturtellerene.

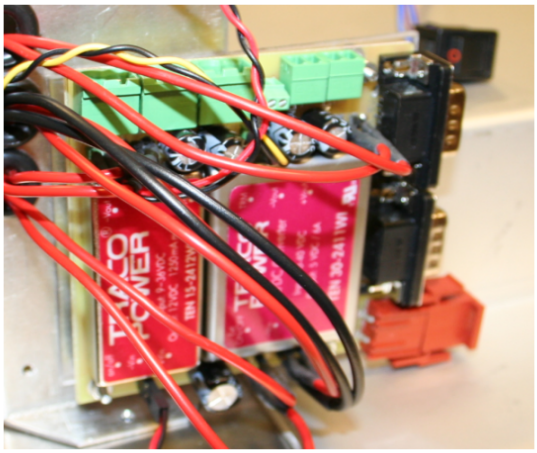
`CAN_ADDR_MOTOR_RIGHT_RETURN_QUAD og CAN_ADDR_MOTOR_LEFT_RETURN_QUAD`
Motordrivermodul som sender en slik melding har mottatt kommandomeldingen `CAN_ADDR_MOTOR_RIGHT_READ_QUAD` eller `CAN_ADDR_MOTOR_LEFT_READ_QUAD`. Meldingen inneholder fire byte, med kvadraturteller verdien siden sist samme melding ble sendt.

4.2.3 Intern Spenningsforsyning

Svitsjet spenningsforsyning

Spenningsforsyningen til modulene på roboten blir tilført spenning fra switch-mode DC/DC konvertere. Det er designet et spenningsforsyningskort som leverer 12V/1250mA, og 5V/6A. Spenningsforsyningen kan operere fra 10 til 36 volt inngangsspenning. Det er lagt vekt på kontaktvalget, slik at det ikke skal være mulig med feilkoblinger. Spenningsforsyningen leverer spenning til alle CAN-modulene på roboten direkte gjennom CAN-buss ledningen. Det er montert to D-Sub 9-kontakter for tilkobling av CAN-Buss. Tre lysdioder er montert, en for indikasjon av tillførselsspenning, samt en til hver av de to utgangene på henholdsvis 5 og 12Volt. I tillegg til spenningsindikasjon fungerer lysdiodene også som belastning for switch-mode-kretsene, som til en hver tid trenger en gitt belastning for å kunne fungere. Sikringer er installert på både inngang og utganger av kortet. Switch-mode-spenningsregulatorene er produsert av Traco Power (TracoPower, 2007), og har typebetegnelsen TEN15WI (TracoPower, 2006a), og TEN30WI (TracoPower, 2006b).

PSU versjon 1.0 er den strømforsyningen som er implementert på roboten. Det er i ettertid gjort små forandringer på 5volt-kontaktene på kretsskjemaet, og designet har fått versjonsnummer 1.1. Kretsdesignet er vist i vedlegg I.9, og



Figur 4.1: Svitsjet spenningsforsyning

bilde av spenningsforsyningen er vist i figur 4.1.

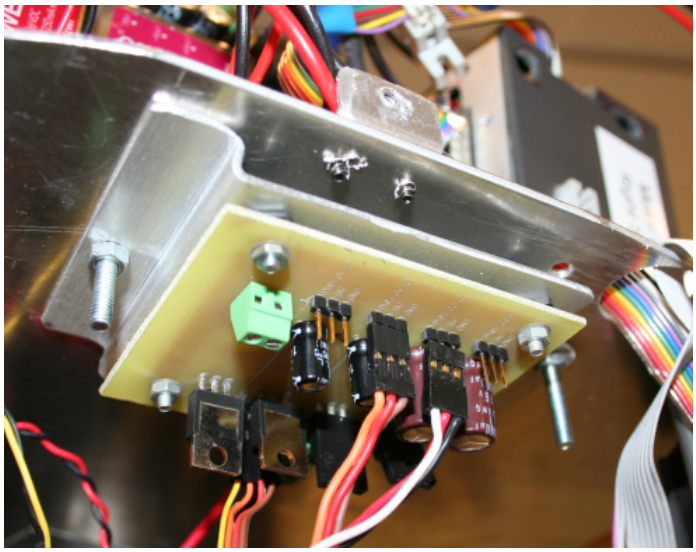
Lineær servo-spenningsforsyning

Til robotens servoer benyttes en egen spenningsforsyning. Dette gjøres for å unngå støy og spenningsfall til resten av robotens elektronikk når servoene belastes. Servo-spenningsforsyningskortet benytter fire lineære spienningsregulatorer av typen L7805 (STMicroelectronics, 2007). L7805 kan belastes med 1.5 ampere, som er tilstrekkelig for de fleste servoer. Dette kortet tar inn pwm signaler på egne kontakter og har standard trepinnsutganger for servoer. Kortet er også ment å kunne brukes som reservekort om det skulle oppstå problemer med 5 volt fra switch-mode spenningsforsyningen. Kretskortet er vist i figur 4.2, og designet er vist i vedlegg I.10.

4.2.4 Antikollisjon

Antikollisjonsystemet på roboten er utviklet av Kristian M. Knausgård. Dette baserer seg på ultralyd og infrarødt lys. Det er montert tre sensormoduler i front av roboten. Her vil grensesnittet sett fra posix-køene og CAN-bussen presenteres. Meldingsidentifikatorene vises med posix-nummeret først, deretter CAN-identifikatoren, hver enkelt melding er presentert med det definerte navnet av verdien fra filene *posixconfig.h* og *proxy_message_config.h*

`MINOR_COLLISION_DETECTED og CANID_COLLISION_DETECTED`
Hver gang det detekteres en kollisjon på en av sensorene, sendes det en CAN-melding med dette nummeret. Systemet er konstruert for åtte sensorer, og sender da en åtte-bytes melding. Hver byte gir avstanden til tilhørende sen-



Figur 4.2: Lineær servo spenningsforsyning

sor detekterer. Er en av sensorene ikke tilkoblet, gir denne sensorplasseringen verdien null. Det sendes kollisjonsmelding hver gang det detekteres en målt avstand under 20 centimeter på en av sensorene. Systemet sender først ut en kollisjons melding når både detektoren med infrarødt lys og ultralyd sensoren detekterer kollisjon samtidig.

`MINOR_RANGE_MEASUREMENTS` og `CANID_RANGE_MEASUREMENTS`
Denne meldingen er en kommando og benyttes for å be om avstanden hver sensor måler i øyeblikket.

`MINOR_FLASH_SENSOR_BOARD_LED` og `CANID_FLASH_SENSOR_BOARD_LED`
Kommandomelding som sendes for å blinke med to lysdioder på hver sensor. Disse lysdiodene blinker hver gang et nytt spill starter.

4.2.5 IO Kort

IO-kortet kan benyttes til en rekke funksjoner. På roboten er det benyttet til å gi meldinger fra tre trykkbrytere. To av bryterne bestemmer hvilken side av bordet roboten starter fra, fargevalg. Tredje bryter benyttes til omstart av programmene for nytt spill. En av tilkoblingene på kortet er tilsluttet en avbruddsinngang INT0. Her kobles robotens startsnor til, se kapittel 2.1. Som en sikkerhet for at et støt eller andre forstyrrelser i roboten under et spill skal aktivere en av bryterne, vil ingen av trykkbryterene på kortet sende meldinger hvis tilkoblingssnoren ikke er satt i. To lysdioder er også implementert på en PWM utgang på kortet. Lysdiodene indikerer lagets farge, og gir en fin tilbakemelding på at planleggings systemet har startet og fungerer.

Det er implementert to kvadraturtellere på IO-kortet, disse kan benyttes til dekoding av enkoderene på løpehjulene, eller som reserve for kvadraturtellerene på motordriverene, se kapittel 4.2.2.

Koden for IO-kortet følger vedlagt på CD under mappen *io_card_with_2quad*. Hele kortet styres ved hjelp av avbruddshåndtering. Avbruddene kommer enten fra trykknappene, startsnoren, eller en mottatt CAN-melding. Etter initialiseringsrutinen *init_IO_card(void)* er kjørt, vil programmet som kjøres ikke gjøre noe før det mottar en avbuddsmelding.

Avbruddshåndtering for trykknappene og startsnoren er implementert i «Interrupt Service Rutines» ISR i filen *interrupts.c*. ISR er avbruddsrutiner som ikke er avbrytbare, det vil si at det ikke kan oppstå nøstede avbrudd. Avbrudd som oppstår når en annen ISR kjøres vil ikke mistes, men kjøres når den aktive ISR rutinen har gjort seg ferdig.

Avbruddshåndtering for CAN-meldinger håndteres i CAN-driveren *at90can128_driver.c*.

Kortet har også utganger og kode for styring av servoer 4.2.8. Denne delen av kortet benyttes ikke på roboten i utgangspunktet, men er ment som redundans for andre servo kontrollere på roboten. Oppstår det problemer med kretskortet til flaske- og boks-sortereren, eller batterisortereren, vil man raskt kunne koble om til å benytte IO kortet.

Grensesnitt IO-kort

Her vil grensesnittet for IO-kortet for Posix-køene og CAN-bussen presenteres. Bare medlinger som er i bruk på roboten presenteres. Kortet har fått adresseområde på CAN-bussen fra 0x130 til 0x13F.

`CAN_INIT_GAME_TIME` og `MINOR_CMD_SYSTEM_INITIALIZE`
Kommandomelding fra IO-kortet som forteller at startsnoren er dradd ut, for start av nytt spill.

`MINOR_CMD_SET_TEAM_COLOR_RED`, `CAN_CMD_SET_TEAM_COLOR_RED`, `MINOR_CMD_SET_TEAM_COLOR_BLUE`, og `CAN_CMD_SET_TEAM_COLOR_BLUE`
Kommandomelding fra IO-kortet om mottatt lagfarge, hennholdsvis rød og blå.

`MINOR_SET_LED_TEAM_COLOR_RED`, `CAN_SET_LED_TEAM_COLOR_RED`, `MINOR_SET_LED_TEAM_COLOR_BLUE`, og `CAN_SET_LED_TEAM_COLOR_BLUE`
Kommandomelding fra planleggingsystemet om å skru på lysdioden tilknyttet IO-kortet som markerer lagets farge.

`CAN_ADDR_INIT_DUAL_QUAD`
Melding som initialiserer kvadraturtellerene på IO-koret.

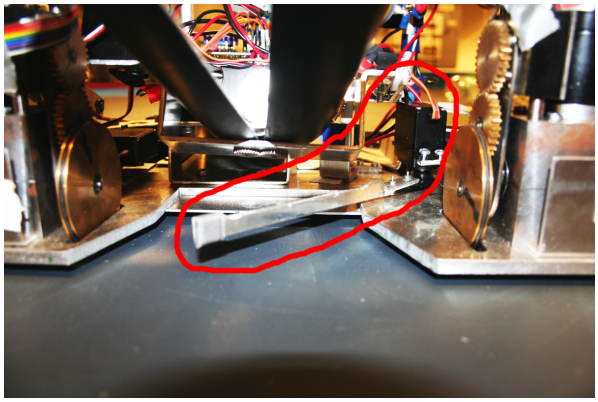
`CAN_ADDR_READ_DUAL_QUAD`
Kommandomelding som ber om utlesning av kvadraturteller verdier.

`CAN_ADDR_RETURN_DUAL_QUAD`
Melding som returnerer en 8 bytes melding med kvadraturteller verdien til de to kvadraturtellerene på kortet. De fire første bytene er fra kvadraturteller en og de fire neste for den andre.

4.2.6 Flaske-/Boks-sorterer

Denne ble utviklet av Eksperter i Team-gruppen, vårsemesteret 2007. Sorteringsenheten ble overtatt og montert inn i roboten en og en halv uke før konkurransen. Etter å ha testet systemet viste det seg at det ikke fungerte tilfredsstillende.

Det første problemet var flasker og bokser som ble liggende under løftearmene når disse var i hevet skyteposisjon. Når en flaske eller boks var skutt og løftearmene skulle senkes ned, ville ikke nedsenking være mulig om det lå en boks, flaske eller et batteri under armene. Problemet her ble løst ved hjelp av en servo tilknyttet en arm som skulle skyve bort flasker, bokser og batteri før senking av armene. Undertegnede samarbeidet med Jan Magnus G. Farstad fra EiT-gruppen på det mekaniske med denne armen. Implementasjon av styreprogrammet ble gjort av undertegnede. Bilde 4.3 viser plassering av armen og servoen den benytter.



Figur 4.3: Oppryddingsarm

Skytemekanismen viste seg heller ikke å fungere tilfredsstillende, da denne hadde en tendens til å stoppe etter skyting, uten å trekke fjæren tilbake. Undertegnede gikk gjennom koden for denne uten å avdekke feilen, for deretter å skrive om koden til å utføre bare de nødvendigste funksjoner. Samme feil ble observert igjen, så sannsynligheten for at feilen lå i koden var nå sterkt

redusert. Det ble studert på hva andre årsaker som kunne forårsake dette problemet, gjennom å studere kode og hendelsesforløpet før enheten stoppet. Når koden ble gjennomgått ved hjelp av Atmels debuggings program AVR Studio 4 (Atmel, 2007), ble det observert at alle brytere som skulle aktiveres for stopp ble aktivert. Dette skjedde like etter at fjæren ble løst ut og det ble skutt. Det var da naturlig å studere bryterene i systemet, og endebryteren for henting av fjæren ble løsnet fra. Bryteren ble deretter aktivert for hånd etter skyting, nå fungerte skytemekanismen som tiltenkt. Feilen viste seg å oppstå ved at skytemekanismens stopper ga et kraftig slag i hele modulen. Slaget aktiverte da endebryteren. Programmet hadde således fått alle signalene den trengte for å stoppe. Det ble gjort to forsøk på å rette feilen. Det første var å bytte bryteren med en mer solid bryter, noe som viste seg å ikke fungere nevneverdig bedre. Deretter ble bryteren flyttet til braketten som holder skytemekanismens kretskort. Dette er lokalisert like over skytemekanismen. Problemet var nå løst, og det oppsto ikke senere problemer med dette.

Heller ikke fjæropptrekket viste seg å fungere tilfredsstillende, da dette av og til gikk for langt og låste seg i endepunktene. Slingringsmonnet på hvor fjæren måtte stoppe var for lite for den mekaniske konstruksjonen, da motoren ikke alltid stopper momentant og på samme plass hver gang. Det var mulig å stoppe fjæren et stykke før, så det ble lagt inn en gummiforing på 0,8 millimeter før stoppunktet etter skyting. Gummiforingen avlastet også den mekaniske konstruksjonen, slik at slaget i denne ikke ble like kraftig som tidligere.

Det ble også observert en del kaldloddinger på kortet som gjorde at det av og til ikke virket. Undertegnede loddet over hele kortet på ny, og tilsvarende problemer har ikke oppstått siden.

Før avreise ble det laget et nytt kort for skytemekanismen av kybernetikk studentene i Eksperter i Team-gruppen. Dette kortet skulle fungere som et reservekort om det skulle oppstå problemer, med det som var montert i roboten under konkurransen.

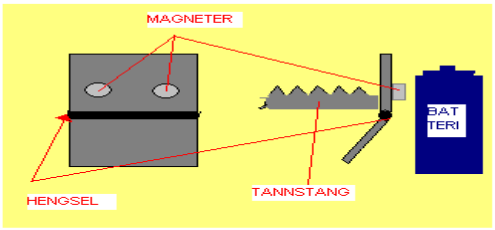
4.2.7 Batteri sorterer

Omtrent halvveis i vårsemesteret 2007, viste det seg at Eksperter i Team-gruppen ville ha nok med flaske-/boks-sortereren, for å få denne til å fungere tilfredsstillende. Det var klare indikasjoner på at batterisortering var lagt på is. Undertegnede besluttet da å designe batterisortereren.

Det ville ikke være stor plass igjen inne i roboten til batterier, og av denne grunn ble det besluttet å plukke ett og ett batteri. Batteriene er magnetiske, noe som gjorde det hensiktsmessig å plukke batterier ved å kjøre på disse med en plate som har magneter i front av roboten. Magnetene måtte være små og sterke, og valget falt da på Neodym-magneter. Magnetene som ble valgt har en diameter på en centimeter, og har en løftekraft på 0,8 kilo. (ClasOhlson, 2007).

Prinsippet som ble valgt for å plukke opp batterier, var å kjøre på batteriene

med en 5x5 centimeter plate i front av roboten, foran venstre motormodul. Platen er festet i en tannstang som kan kjøres inn og ut av roboten ved hjelp av en motor. Når tannstangen kjøres ut av roboten løftes den og frontplaten med 15 graders vinkel fra underlaget. Når tannstangen kommer i ytre stilling senkes hele tannstangen ned slik at frontplaten kommer en centimeter over spillebordet. Vinkelen sammen med senkingen er nok til at batterier som er festet i frontplata blir løftet over kanten på batterisorteringsboksen, og står nå oppi boksen. Frontplata har da et hengsel på midten som kan vippes utover fra roboten, samt at to magneter er festet på oversida av denne hengselen. Når roboten har senket batteriet ned i sorteringsboksen, kan den rygge tilbake, og batteriet vil falle av ved hjelp av den hengslede delen av plata, som vil presse av batteriet. Prinsippet vises forfra og fra siden i figur 4.4.

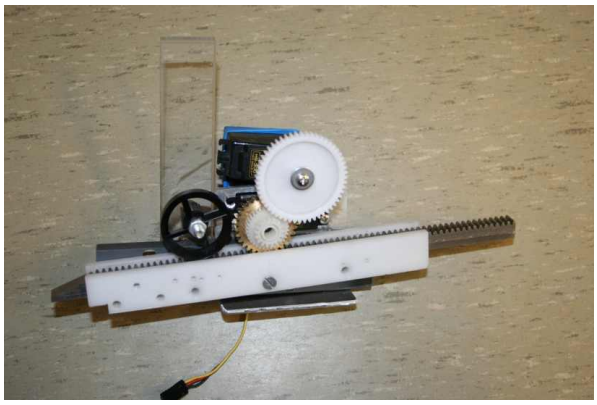


Figur 4.4: Prinsipp for batterisorteringsplate

En modifisert servo ble valgt til motor for denne enheten, da den skal gå inn og ut av roboten, og like langt hver gang. Hobbyservoer kan beveges en vinkel på 180 grader, der de i endepunktene har fysiske stoppere. Regulatoren i servoen har et potensiometer som den bruker til vinkelmåling. For å kunne benytte en hobbyservo, var det nødvendig med modifisering av denne, da tannhjulet som gikk inn på tannstangen måtte rotere to ganger for å kjøre denne 10 centimeter ut. De fysiske stopperene i endepunktene ble fjernet, slik at servoen kunne rotere fritt. Tilbakekoblingen fra potensiometeret ble løst ved å benytte et ekstra servopotensiometer. Det ekstra servopotensiometeret ble giret fire ganger for å gi ønsket utslag for tannstangen.

Vippemekanismen for tannstangen ble løst ved å skråskjære tannstangen i bakkant, samt å la et hjul holde tannstangen nede i bakkant. Når tannstangen kommer så langt frem at den er nesten i ytre posisjon, vil senking skje når støttehjulet kommer til denne skråskjæringen.

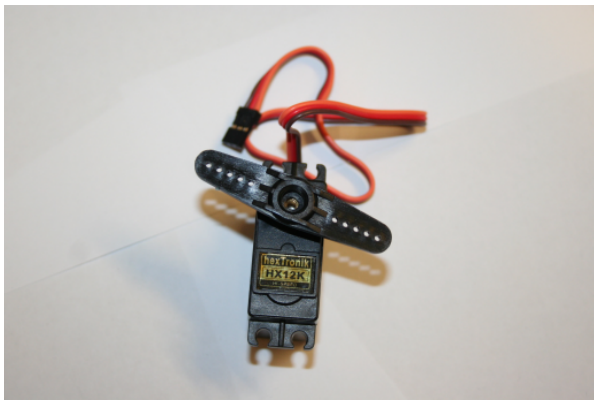
Bilde av batterisorteren uten frontplata er vist i figur 4.5.



Figur 4.5: Batterisorterer

4.2.8 Servoer

Hobbyservoer finnes i mange utførelser. De fleste hobbyservoer er produsert med plastgir, som har lav bruddstyrke. Det var derfor viktig å benytte servoer som tåler stor belastning hvis det mekaniske systemet skulle låse seg. Valget falt av denne grunn på servoer med metallgir. Denne typen servo har en høy pris i Norge, og ble bestilt fra Kina (UnitedHobbies, 2007) for å spare penger. Servoen som ble benyttet var hexTronik HX12K. Bilde av en slik servo er vist i figur 4.6.



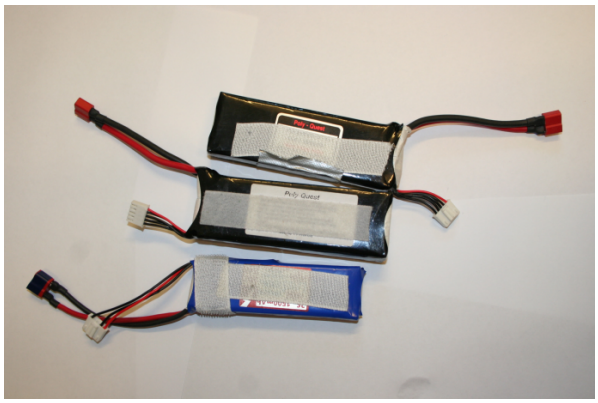
Figur 4.6: Servo med metallgir

4.2.9 Batterier

For robotens spenningstilførsel blir det benyttet tre batterier av typen Lithium ion polymer, som refereres til som LiPo batteri. LiPo batterier tar liten plass og har lav vekt i forhold til effekten de kan levere. Dette er en fordel, da man har begrenset plass tilgjengelig i roboten.

Et batteri på 14,8 volt blir benyttet til hovedkort og spenningstilførsel til swich-mode-spenningsforsyningen, og et tilsvarende batteri på 14,8 volt til de tre motorene i roboten. De to 14,8 volts batteriene har en kapasitet på 2200mAh. Batteri til servospenningsforsyningen, består av et 7,4 volts batteri på 1500mAh. Bilde av batteriene er vist i figur 4.7.

Batteriene til motorene og servoene kan benyttes på roboten i flere timer uten lading, da disse ikke har kontinuerlig belastning. Hovedkortet sammen med modulene som drives av switch-mode-spenningsforsyningen trekker fra 1,5 til 2,5 Ampere, noe som gir roboten en kjøretid mellom lading på omtrent en time.



Figur 4.7: Robotens Batteri

Inne i roboten festes batteriene sammen med borrelås, for raskt og enkelt å kunne bytte disse. Det er montert en brakett over venstre bakhjul for plassering av batteriene.

4.2.10 Kontakter

Tilkoblingspunkter for spenning- og signal-ledninger ble viet en del oppmerksomhet under konstruksjonen av roboten. Det ble lagt vekt på å skulle kunne koble til de forskjellige delene av roboten uten at man skal kunne koble feil med tanke på polaritet og tilknytningspunkt. Løsningen ble å velge en del forskjellige typer kontakter, samt forskjellige farger på disse. Viktige kriterier var og at det skulle være lett å koble inn og ut enheter og moduler, slik at alle moduler måtte ha kontakter og ikke skrutilkoplinger for ledninger.

4.3 Programvare

4.3.1 Operativsystemet

Operativsystem for roboten ble valgt under prosjektperioden 2006 (Kjemphol, Gunnar og Knausgaard, Kristian M, 2006). Server-versjonen av Ubuntu Linux ble valgt på grunnlag av gode erfaringer fra Eurobot 2006 roboten, samt årets Eurobot deltageres kjennskap til operativsystemet. Linux er regnet som et stabilt operativsystem for innebygde datamaskinsystemer.

Secure shell «SSH» er installert på operativsystemet. Ved hjelp av SSH kan flere personer jobbe på roboten samtidig. Man er heller ikke avhengig av å ha skjerm og tastatur tilkoblet på hovedkortet.

Installasjonsprosedyren for operativsystemet på hoveddatamaskinen og Compact Flash kort er vist i vedlegg B.

4.3.2 CAN-buss

CAN-buss ble valgt som kommunikasjonssystem mellom hoveddatamaskin og de eksterne noder under prosjektperioden høsten 2006. Teori på denne finnes i prosjektrapporten (Kjemphol, Gunnar og Knausgaard, Kristian M, 2006). CAN-buss ble valgt ut fra ønsket om en mest mulig modulbasert robot, med mest mulig gjennbrukbare deler. Et annet kriterium som sto høyt ved valg av CAN-buss, var muligheten for på en enkel måte å bytte ut moduler som viste seg å ikke tilfredsstille de ønskede krav.

Det ble i prosjektperioden utviklet en paralellport til CAN kontroller, for komunikasjon mellom hoveddatamaskinen og resten av robotens CAN-noder (Kjemphol, Gunnar og Knausgaard, Kristian M, 2006). Kontrollerens hardware ble ferdigstilt, og en enkel driver ble utviklet for Linux. Det var ikke implementert feilhåndtering i denne driveren, og driveren var ikke testet over lang tid. For å spare utviklingstid av driver til denne, ble det valgt å benytte ferdigkjøpt kontroller som tidligere bare var benyttet til debugging av det utviklede utstyret. Et annet kriterium som ble vektlagt for den kjøpte kontrolleren, var sannsynligheten for feil i en egenutviklet driver, ville være betydelig større en en utprøvd kommersiell driver. CAN-kontroller som ble benyttet var da en USB til CAN kontroller (PEAK, 2007a) fra Peak System. CAN kontrolleren har USB grensesnitt mot hoveddatamaskinen. Peak har tilgjengelig Linux driver for denne kontrollernen, på sin hjemmeside (PEAK, 2007b). Installasjonsveiledning for driveren er vedlagt i vedlegg C. I proxy-/gateway-systemet 4.3.7 er initialisering og bruk av driveren gjort i filen *can_buss.cpp*. Bilde av CAN-kontrolleren er vist i figur 4.8

Etterhvert som flere moduler kobles til roboten, øker antall meldinger på CAN-bussen. Enhver CAN-buss-modul har mulighet til å filtrere meldinger. Filtreringen benyttes på identifikatoren, slik at hver node kan ta imot meldinger

Figur 4.8: PCAN USB-CAN-kontroller

med bare de relevante identifikatorer. Filtreringen skjer i hardware på CAN-kontrolleren. Uten meldingsfiltrering ville hver node måtte sjekke i software ved hjelp av polling eller avbruddshåndtering om meldingen er relevant eller ikke.

Atmel AT90CAN128

Alle eksterne noder på roboten benytter Atmels mikrokontroller AT90CAN128 (Atmel, 2006), som har innebygget CAN-kontroller. Kontrolleren har 15 meldingsbokser nummerert fra 0 til 14. Disse meldingsboksene kan konfigureres til en av følgende oppsett: Deaktivert, Meldingssending, Meldingsmottak, Automatisk svar, eller Buffer meldingsmottak. Hver av meldingsboksene må konfigureres for seg.

Mottak og filtrering av meldinger

Meldingsboksene som er satt opp for mottak, kan filtrere på hvilke meldingsidentifikatorer de ønsker å motta. Filteret i meldingsboksen settes opp til å motta alle meldinger i et gitt område, da altså fra en identifikator til en annen identifikator. Ønsker man å motta meldingsidentifikatorer i to områder, for eksempel fra 0x90 til 0x9F og fra 0x110 til 0x11F, må man sette opp to meldingsbokser for mottak, hvis meldingsfiltrering skal benyttes.

Følgende kode setter opp meldingsfilter for meldingsboks 0:

```
1  #define CONMOB_DISABLE 0x00
2  #define CONMOB_ENABLE_RESEP (1 << CONMOB0)
3
4
5
6  // Chanel 0 MOb page 0 enable for recieving
7  CANPAGE = (0 << 4);       // Message box 0 / MOB 0
8  CANSTMOB = 0x00;          // Clear MOb Status register.
9  CANCDMOB = CONMOB_DISABLE;// MOb Configuration
10 CANIDT1 = 0x22;           // Identifier TAG (IDT10:0) 11 bits.
11 CANIDT2 = 0x00;           // From 0x110 to 0x11F
```

```
12
13 CANIDM1 = 0xFE;           // Filter
14 CANIDM2 = 0x00;           // 11 1111 0000
15 CANIDM4 = 0;              // comparison true forced
16
17 CANCDMOB |= 0x08;         // Reception 8 bytes
18 CANCDMOB |= CONMOB_ENABLE_RESEP;
19
20 CANIE2 |= 0x01;
21 CANGIE = (1<<ENRX);       // Recieve enable
```

Som man ser av koden legges laveste identifikatoren man ønsker mottatt i gjeldende meldingsboks inn i registrene $CANIDT1$ og $CANIDT2$. Deretter settes filteret som bestemmer hvor høyt opp i identifikatorvedi man ønsker å motta meldinger i registrene $CANIDM1$ og $CANIDM2$. Videre må filteret aktiveres ved å sette registeret $CANIDM4 = 0x00$. Når en melding er mottatt i en meldingsboks gir CAN-kontrolleren beskjed til mikrokontrolleren ved å sette et flagg. Mikrokontrolleren settes opp med avbruddshåndtering på dette flagget med de to siste linjene i koden, registrene $CANIE2$ og $CANGIE$.

Meldingssending

For oppsett av en meldingsboks for sending benyttes følgende kode, her for meldingsboks nummer 1:
I initialiseringrutinen må følgende kode kjøres:

```
1 // Channel 1 MOb page 1 enable for transmitting
2 CANPAGE = (1 << 4);
3 CANSTMOB = 0x00;                  // Clear MOb Status register.
4 CANCDMOB = CONMOB_DISABLE;        // MOb Configuration
5
6 CANGIE = (1<<ENIT);               // Transmit enable
```

Før man sender en ny melding må en del data legges inn i meldingsboksens registere:

```
1 CANCDMOB = CONMOB_DISABLE;
2 CANPAGE = (1 << 4);               // Select chanel 1
3
4 CANIDT2 = 0x00;
5 CANIDT1 = 0x0E;                   // 0x70
6
7 CANCDMOB |= 0x02;                 // Data length code DLC
8 CANMSG = DATABYTE1;
9 CANMSG = DATABYTE0;
10
11
12 CANCDMOB |= CONMOB_ENABLE_TRANS;// Enable transmission
13 CANGIT = CANGIT;                 // Reset all flags
14 CANSTMOB = 0x00;                 // Clear MOb Status register.
15 CANCDMOB = CONMOB_DISABLE;       // MOb Configuration
```

Her velger man hvilken meldingsboks man ønsker å sende fra ved å velge ønsket meldingsboks i registeret $CANPAGE$. I registrene $CANIDT1$ 0g $CANIDT2$ legges identifikatoren til meldingen man ønsker å sende. For koden her er det identifikator 0x70. Lengden på meldingen settes i $CANCDMOB$, før meldingsdata legges fortløpende inn i registeret $CANMSG$. Deretter settes $CANCDMOB$ til sendingsmodus, og meldingen sendes.

4.3.3 Posix-kø systemet

Meldingssystemet som er benyttet på roboten mellom de forskjellige programmene på hoveddatamaskinen, kommuniserer ved hjelp av posix-køer. Systemet er utviklet av Kristian M. Knausgård våren 2007. Det blir her gitt en gjennomgang i bruken av systemet for å få en helhetlig forståelse av hvordan roboten fungerer, og for å gi mulighet til benyttelse av systemet.

Systemet benytter følgende filer: $MessageHandler.h$, $MessageHandler.cpp$, $posix\_message\_config.h$, og $posix\_message\_queue.c$.

Man benytter systemet ved å opprette et MessageHandler objekt på følgende måte:

$MessageHandlerplansysMessageHandler(PROXY\_ID,$
$PMQ\_PROXY, PMQ\_PLANSYS, true);$

Her er første argument i opprettelsen av objektet $PROXY\_ID$, en variabel som benyttes for debugging. Denne forteller her at det er PROXY systemet som sender meldingen. De to neste argumentene definerer inn-køen og ut-køen, altså her er det PMQ_PROXY man mottar meldinger på og $PMQ\_PLANSYS$ man sender meldinger til. Siste argument settes $true$, eller $false$, $true$ betyr at meldingssystemet er avbruddsdrevet, og ved $false$ må man selv sørge for å sjekke om der er nye meldinger i køen, som venter på å bli lest.

Meldinger for systemet benytter MAJOR- og MINOR-nummer. Disse nummerere forteller hva type melding det er som er mottatt eller sendes til køen. MAJOR nummeret forteller om meldingen er en kommando uten data, eller en strukt med data. MINOR-nummeret beskriver oppgaven til melding som er sendt og hvilken funksjon som skal kjøres, når dette MINOR-nummeret er mottatt.

Mottak av meldinger skjer ved at det først må opprettes en SubMessageHandler på MessageHandler-objektet. Her legges det til en peker til en metode som behandler den mottatte meldingen som har tilhørende MAJOR og MINOR nummer.

$plansysMessageHandler.addSubMessageHandler(RecieveCommand,$
$MAJOR\_COMMAND, MINOR\_MESSAGE);$

Funksjonen *RecieveCommand* blir nå kjørt hver gang det kommer en melding med MINOR verdi MINOR_MESSAGE. MINOR-verdiene er definert i filen $posix\_message\_config.h$. Alle funksjoner som blir kjørt når en ny melding ankommer, tar inn en peker til data i meldingen. For en melding med MAJOR-nummer MAJOR_STRUCT vil pekeren peke på tilhørende stukt. For en kommando vil den peke til NULL, altså melding uten data.

Sending av meldinger ved hjelp av posix systemet benytter følgende kommando:

$$plansysMessageHandler.sendMessage((char*)\&message\_stuct, (int)sizeof(message\_stuct), MINOR\_MESSAGE, PRIORITY)$$

Her er første argument en peker til strukten som ønskes sendt, neste argument er størrelsen på strukten. De to siste argumentene gir MINOR nummeret for meldingen, og prioriteten meldingen skal ha i posix køen.

4.3.4 Strategiplanleggingssystem

Det var ønskelig å ha et strategiplanleggingssystem for roboten som skulle utføre tiltenkt oppgave under alle omstendigheter. Det er lagt ned mye arbeid for å finne riktig type strategiplanleggingssystem. Etter nøye gjennomgang av flere mulige løsninger, sto valget mellom to forskjellige systemer som kunne være aktuelle for årets robot. Det første var å implementere en stor tilstandmaskin med en fast strategi. Den andre aktuelle løsningen var å implementere et «action selection» system, der en algoritme til en hver tid velger ut en strategi basert på de forskjellige strategiers momentane prioritet.

For roboten falt valget av strategiplanleggingssystem på «action selection»-metoden og tankegangen til Pattie Maes. Maes «action selection»-algoritme kombinerer karakteristiske trekk fra både tradisjonnelle planleggingssystem i det hierarkiske paradigmet 3.1.1, og reaktive systemer 3.1.2, (Pattie Maes, 1989).

Det ble søkt en del på Internett etter ferdig rammeverk implementert i C eller C++ for denne type oppgave, men ingen ting av interesse ble funnet. Det ble da bestemt å implementere et slikt system fra bunnen av.

Ønskede kriterier for et strategiplanleggingssystem er som følger:

- Man skal ha mulighet for et stort antall forskjellige uavhengige strategier.
- Prioritere målorienterte strategier.
- Man skal velge strategi ut fra hvilken strategi som har høyest prioritet.
- Hver strategi skal ha mulighet til å ha all tilgjengelig oppdatert informasjon til en hver tid.

- Robust, ved at den kan fjerne strategier som ikke virker.
- Må kunne håndtere avbrudd uten at dette påvirker aktive strategier.
- Aktiv strategi skal beholdes til den har oppnådd sitt mål, eller en annen strategi gir betydelig høyere prioritet.

Et av kriteriene for å designe et eget strategiplanleggingssystem var et gjenbrukbart rammeverk. Ser man på et rammeverk som en trestruktur der rammeverket er roten og stammen, er det viktig at dette er gjennomtenkt. Om man ved en senere anledning oppdager en fundamental feil i det grunnleggende rammeverket, kan det få store konsekvenser for alle løvnoder som er koblet på systemet. Er rammeverket gjennomtenkt og det er løvnodene som inneholder feil, kan disse feilene raskt rettes uten at det påvirker resten av systemet (Brown W., Malveau R., McCormick H., Mowbray T., 1998).

Action selection strategiplanleggingssystem

Det implementerte strategiplanleggingssystemet på roboten er implementert rundt design patteren Observer (Gamma, E. ,Helm, R. , Johnson, R. ,Vlissides, J., 1995). Ved hjelp av «Observer design pattern» vil alle nye meldinger som kommer inn til planleggingssystemet, bli sendt til, og oppdatere alle strategier. Man kan raskt legge til og fjerne strategier fra systemet, da en peker til alle strategier som skal kunne brukes ligger i en vektor-liste.

Hver strategi i systemet er implementert som egne uavhengige objekter, som igjen kommuniserer mot ett objekt *PlansysObservation*. Dette velger ut gjeldende strategi. Ved å benytte denne oppbyggingen av systemet, kan man legge til og fjerne strategier både når programmet kjører, og i koden uten å måtte forandre på rammeverket og de strategier som allerede er implementert. Systemet er dynamisk slik at det ikke er begrenset hvor mange strategier man kan ha. Det er også mulig ved hjelp av en variabel i hvert strategi-objekt å sette en strategi inaktiv.

Strategiene i systemet kommuniserer gjennom to strukter. Den første strukten *controll_struct* inneholder den aktive strategiens handlinger, og kommandoer, som nye «waypoints», og meldinger til systemet. Den andre strukten *interrupt_controll_struct* inneholder avbruddshandlinger, som alle strategier til en hver tid kan sette.

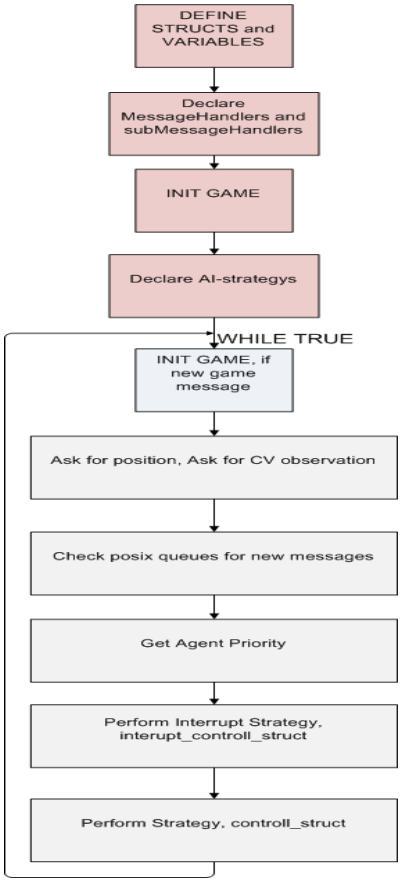
Systemet er bygd opp slik at når en strategi først er valgt, vil denne settes aktiv. Den aktive strategien beholdes til dennes oppgave er utført eller blir avbrutt og nullstilles. En strategi kan avbrytes av en avbruddsmelding fra en annen strategi, gjennom objektet *PlansysObservation*. Utvelgelsen av en strategi gjøres i metoden *GetAgentPriority*() i klassen *PlansysObservation*, deretter utføres den valgte strategien ved hjelp av metoden *PerformStrategy*() i samme klasse.

Planleggingsystemet kommuniserer med resten av roboten gjennom posix meldingskø systemet 4.3.3, men er ikke låst til å bruke dette. Systemet kan

benyttes mot et hvilket som helst meldingssystem, da hele grensesnittet for kommunikasjon med omverdenen ligger i filen *mainplansys.cpp*.

Main-metoden

Main-metoden til planleggingsystemet er bygget opp med først en rekke initialiseringsmetoder, for deretter å kontrollere roboten fra en while løkke. Blokkdiagramet figur 4.9 viser de forskjellige blokkene av main-metoden.



Figur 4.9: Main-metoden

Blokkene starter med å definere strukter og variable programmet benytter, før opprettelsen av MessageHandler-objektene for de forskjellige posix-køene, defineres i neste blokk. Meldingshåndterere for posix-meldingene opprettes også i blokk to. Initialisering av nytt spill kjøres i blokk tre. Før while-løkken i hovedprogrammet, opprettes alle strategiene som skal benyttes. Deretter kjører programmet i en evig while-løkke.

While-løkken starter med en blokk som kan starte et spill på nytt. Blokken er aktiv bare om en initialiseringsmelding har kommet fra en posix-kø.

Første programmet gjør i while løkka etter å ha sjekket for initialiseringsmelding er å be om posisjonsoppdatering og bildeoppdatering. Bildeoppdatering spør roboten om i det implementerte programmet bare når det ikke er aktive strategier. Deretter sjekkes posix-køene for meldinger, alle meldinger som ligger i disse køene blir lest ut, og strategiene blir oppdatert med informasjonen.

Prioritetsforespørsel blir sendt til alle strategier, hvor på de to siste blokkene utfører henholdsvis avbruddshandlinger og vanlige strategi handlinger.

Tid

Tid er viktig å holde rede på for planleggingssystemet, da alle kampene går på tid. Det skal også være mulig for de forskjellige strategier å planlegge handlinger ut fra hvor mye tid som er igjen av spillet. Tidsovervåkingen kjøres i en egen tråd i systemet og initialiseres ved hjelp av følgende funsjon:

```
/**
 * Init and Start the timer thread
 */
pthread_t thread1;
pthread_create(&thread1, NULL, timer_thread, &
    proxysysMessageHandler);
```

Tidstråden tar inn en peker til proxymeldingshåndtereren, slik at den kan sende melding til proxysystemet. Det sendes melding når nytt spill startes og når spilltiden er over.

En spill-resett knapp er implementert på roboten, 4.2.5. Denne er implementert på IO-kortet og skal gi beskjed til planleggingsystemet om nytt spill. Mottar roboten meldingen posix-meldingsidenitfikatoren:
$MINOR\_CMD\_SYSTEM\_INITIALIZE$, vil flagget
$gameResetMessageTimer$ bli satt til $TRUE$. Da vil tidstråden stoppe og vente på melding om lagets farge, samt spill-startmelding fra IO kortet. Når startmelding mottas vil planleggingssystemet igjen sende initialiseringsmelding til resten av systemet. Etter initialisering kjører tidstråden i en while-løkke. Tidstråden sjekker hvor mye tid det er igjen cirka hvert millisekund, og oppdaterer startegiene hvert tiende millisekund.

Funksjonen
$plansysObs.NewObservation(pos\_mes, GAME\_TIME\_MESSAGE)$
brukes for å oppdatere tiden i strategiene.

Når tiden er ute sendes melding til proxysystemet om å stoppe alle aktuatorer som Eurobot reglene krever (Eurobot Commitee, 2006).

PlansysObservtion-klassen

Utvelgelsen av den aktive strategien for planleggingssystemet utføres av metoden *GetAgentPriority* i *PlansysObservation*-objektet. I denne metoden

kjører en algoritme som velger den strategien med høyest prioritet. Når algoritmen har valgt ut en strategi, vil denne strategien være aktiv til den har gjort seg ferdig eller blir avbrutt. Når det kommer en ny observasjon til planleggingsystemet fra posix køene, bli funksjonen *NewObservation* kjørt. Funksjonen *NewObservation* kjører igjen $Notify$, som er en metode som ligger i *Subject*-klassen *PlansysObservation*-klassen er arvet fra. $Notify$ benytter Vector-listen som inneholder peker til alle strategiene, og kjører *Update*-funksjonen til alle strategiene. Når *PlansysObservation* kjører *Update*-funksjonen på strategiene, sender den med en peker til seg selv og hvilken type observasjon som skal oppdateres. Pekeren benytter strategiene til å hente gjeldende observasjon fra *PlansysObservation*.

Hver ny observasjon i PlansysObservtion-klassen har en egen metode som returnerer en strukt med gjeldende observasjon.

Strategiene

Strategiene er implementert som egne objekter, og nye strategier kan implementeres ut fra behov. Det eneste som er fastsatt er måten de kommuniserer på gjennom de to struktene, og en del fastsatte metoder. En stor fordel med å implementere strategiene som egne objekter, er at om det oppstår en feil i en strategi som påvirker en observasjon, vil denne feilen ikke påvirke de andre strategiene, da alle observasjoner kopieres inn i egne strukter i hver strategi. Eventuelle andre strategifeil vil heller ikke kunne påvirke de andre strategiene.

Man kan gjerne ha flere strategier for flaske, boks eller batterisortering, som er implementert på forskjelige måter. Det kan være strategier som for eksempel gir høyere prioritet ut fra gitte kriterier i spillet, som da for eksempel hvor roboten er på banen, eller hvordan den velger å sortere.

Batterisorteringsstrategien bør gi høy prioritet om den vet at motstanderens robot blokkerer oppsamlingboksene for flasker og bokser, samtidig bør da flaske og boks strategien gi lav prioritet.

Alle strategiene kan benytte A-Star-algoritmen som er utviklet til roboten. Denne algoritmen ligger tilgjengelig og kan benyttes av hver strategi i planleggingssystemet 4.3.4.

Implementasjon av strategiene

For å kunne benytte strategiplanleggingsprogrammet må man implementere strategiobjekter. Strategiobjekter er objekter arvet av klassen Observer. Det er implementert et objekt *EmptyStrategy* som inneholder alt et strategiobjekt må inneholde av variabler og funksjoner som et minimum. Denne strategien har ingen funksjon for roboten, men er ment som en mal for å implementere nye strategier. Hedder-filen for *EmptyStrategy* er her vedlagt i Listing 4.1. De implementerte stragegiene *BottleBoxSort* og *BatterySort*, vil ikke bli gjennomgått i detalj når det gjelder oppbygging, da en gjenomgang av strategiobjektet *EmptyStrategy* vil gjøre de to andre strategiene selvforklarende.

Listing 4.1: EmptyStrategy.h

```
1  #ifndef EMPTY_STRATEGY_H
2  #define EMPTY_STRATEGY_H
3  /**
4   *  Eurobot 2007
5   *  EmptyStrategy.h
6   *  author "Gunnar Kjemphol <gunnar@kjemphol.no>"
7   *
8   *  This is a child class of Observer class
9   *  The PlansysObservation objects can call EmptyStrategy memberfunction
10  *  Update() when a new observation is recieved.
11  */
12
13 #include "Observer.h"
14 #include "PlansysObservation.h"
15 #include "mainplansys.h"
16 #include "waypoint.h"
17 #include "posix_message_config.h"
18 #include "observations.h"
19 #include "init.h"
20
21 class EmptyStrategy: public Observer {
22   public:
23     EmptyStrategy( PlansysObservation * , int theTeamColor );
24     ~EmptyStrategy();
25     void Update( Subject *, int typeOfObservation);
26     int GetPriority();
27     int AgentGlobalPriority();
28     int GetAction( controll_struct *ai_controll_struct );
29     int SetAgentActive( int active );
30     int StrategyInterrupted();
31     int GetInterruptAction(interrupt_controll_struct *interrupt_controll);
32
33   private:
34   /**
35    * Give the Agent a priority in the constructor [0-9]
36    * if the agentPriority is -1 the agent is inactive
37    */
38     int agentPriority;
39     int inAction;
40     int activeStrategy;
41
42     PlansysObservation *_subject;
43     ai_waypoint ai_waypoint_3d;              // Position
44     ai_waypoint ai_waypoint_reached;         // Last waypoint reached
45     cv_observation my_cv_observation;        // Last CV observation
46     cv_observation color_cv_observation;     // CV color observation
47     proxy_message my_proxy_message;          // Proxy Message
48     controll_struct local_controll_struct;   // Control Struct
49 };
50 #endif
```

Når et nytt strategiobjekt opprettes, tar det inn to argumenter. Det første argumentet er en peker til *PlansysObservation* objektet. Det andre argumentet strategiobjektet tar inn er en variabel med lagets farge.

Funksjonen *Update* blir kjørt av *PlansysObservation*-klassen hver gang en ny melding kommer inn til planleggingsystemet. Funksjonen tar inn en peker til *Subject*-klassen som *PlansysObservation* objektet er arvet fra. Den skal benyttes til å hente den nye observajonen fra *PlansysObservation*. Andre argument i funksjonen er en variabel som forteller hvilken type observasjon som er mottatt. Hver strategi bestemmer da om observasjonene er relevant, og kan da hente den nye observasjonen fra *PlansysObservation*-objektet.

GetPriority er en funksjon som skal returnere prioriteten til strategien. Prioriteten gir ut fra gitte forutsetninger som observasjoner osv.

Når en strategi er aktiv, blir funksjonen *GetAction* kjørt på denne. *GetAction* tar inn kontrollstrukten og legger inn i denne de handlinger strategien ønsker roboten skal utføre.

GetPriority og *GetAction* er nært knyttet til hverandre når en strategi er aktiv. På robotens strategier *BottleBoxSort* og *BatterySort* er *GetPriority* implementert som tilstandsmaskiner. Tilstandsmaskinen kan vel så gjerne ligge i *GetAction*, da det bare er grensesnittet for en strategi som er låst.

PlansysObservation-objektet benytter funksjonen *SetAgentActive* til å underette en strategi at denne nå er aktiv strategi.

Ved hjelp av funksjonen *GetInterruptAction* kan en strategi gi avbruddsmeldinger til roboten. Den kan avbryte aktive strategier eller gi kommandoer til deler av systemet. Alle strategier kan legge inn meldinger her selv om de ikke er aktiv strategi. På roboten er det bare *MotorAi*-strategien som benytter seg av denne. Funksjonen tar inn en peker til *interrupt_controll_struct*, og legger kommandoer rett i denne. Dette er eneste strukten som ikke strategiene har en egen lokal variant av.

BottleBoxSort Strategien
Strategien er implementert med en tilstandsmaskin i metoden *GetPriority*. Har Strategiobjektet fått melding om flaske eller boks observasjon foran roboten, vil denne strategien gi høy prioritet. Blir strategien valgt av roboten, vil resten av tilstandsmaskinen bli kjørt til den er ferdig med oppgaven eller blir avbrutt.

Strategien kommuniserer mot navigasjonssystemet og flaske-/boks-sortereren, gjennom strukten *controll_struct*. Grensesnittet mot de to modulene er beskrevet i henholdsvis navigasjonssystemet 4.3.5, og under beskrivelsen av flaske-/boks-sortereren 4.2.6.

BatterySort Strategien
Batterisorteringsenheten ble som flaske-/boks-sorterern implementert som en tilstandsmaskin. Strategien får høy prioritet om bildebehandlingssystemet har gitt beskjed om batteriobservasjon. Grensesnittet for batterisortereren kommu-

niserer mot navigasjonssystemet og IO-kortet hvor koden for batterisortereren er implementert.

MotorAi Strategien
MotorAi strategien tar seg av meldinger fra antikollisjonsystemet og styrer roboten unna hindringer. Grensesnittet mot antikollisjonssystemet er beskrevet i kapittel 4.2.4. Strategien kommuniserer gjennom strukten *interrupt_controll_struct*, og setter avbrudds-waypoint som blir kjørt idet navigasjonssystemet mottar disse, se kapittel 4.3.5. Strategien har også mulighet til å stoppe og starte roboten. Funksjonen *GetPriority* returnerer -10, slik at denne strategien aldri blir valgt som aktiv strategi.

Når strategien mottar en melding fra antikollisjonssystemet mottar den avstandsmålig fra alle sensorer som er tilkoblet. Slik systemet er implementert begynner strategien å sjekke på venstre sensor, har den detektert en kollisjon, vil strategien sette et avbrudds-waypoint 35 centimeter rett til høyre. Er det derimot kollisjon på høyre sensor og ikke venstre, vil et avbrudds-waypoint 35 centimeter rett til venstre bli satt. Detekteres det kollisjon på den midterste sensoren, vil det settes et avbrudds-waypoint som snur roboten og kjører den 50 centimeter og 45 graders vinkel bort fra kollisjonen.

Man kan implementere mange varianter av unnamanøvere i denne strategien. Man kan også ta høyde for hvor lenge det er siden forrige detekterte kollisjon, før man utfører en handling.

FirstStrategy
Strategien er implementert for å gi høy prioritet de første 10 sekundene etter at et nytt spill er startet. Strategien benyttes da til å kjøre en rute for å komme til andre siden av bordet. Ruten velges ut fra plassering av batterisorteringsboksen. Etter at de 10 sekundene er gått, vil strategien gi negativ prioritet og da ikke lenger kunne settes aktiv.

Kognitiv Aktivitet

Som beskrevet i kapittel 3.1.2 er det å bestemme hva det skal letes etter en kognitiv aktivitet. Planleggingsystemet er designet slik at det spør om observasjoner fra bildebehandlingssystemet når det ikke jobber med en spesiell strategi.

A-Star

Justin Heyes-Jones har siden 2001 drevet en internettside for undervisnings formål på A-Star algoritmen (Heyes-Jones, 2007b). I 2006 opprettet Heyes-Jones et prosjekt for A-Star algoritmen på Google Code (Heyes-Jones, 2007a) i tillegg til den opprinnelige internettsiden. Her er algoritmen implementert i C++, og er utgitt på fri benyttelses-lisens, eneste restriksjon er at filen *licence.tex* følger

med der koden benyttes.

A-Star koden er implementert på roboten, og alle strategiene kan benytter seg av denne. Undertegnede har valgt å benytte koden fra prosjektet til Heyes-Jones. Koden er forandret, slik at den nå opererer med waypiont basert på strukten *ai_waypoint* som planleggingsystemet benytter.

På vedlagt CD følger et ekempelprogram som viser bruken av koden. Eksempelprogrammet finnes under $Kode- > Heyes - Jones\_astar\_demo$. Det kjørbare testprogrammer ligger i prosjektmappen under katalogen *src*, hvor den kjørbare filen heter *astartest*. For å benytte testprogrammet må man først ha en grid-verdensrepresentasjon, gitt i X og Y koordinater, som legges inn i ved hjelp av metoden $InitMap$. I filen $AStarUse.h$ kan største grid som skal kunne benyttes settes, i testprogramet er X satt til 10 og Y til 15. Før man begynner å søke etter en rute i nodene, må man initialisere rutenettet, og legge inn vekting på nodene. Initialiseringen kjøres av funksjonen $InitMap(x, y)$, som setter alle nodene til verdien 1. X og y verdien som sendes med $InitMap$ gir hvor stor grid representasjon som ønskes brukt. Vektene kan i programmet settes ved hjelp av funksjonen $SetMapWeight(x, y, weight)$, og til verdier fra 1 til 9, hvor 1 er minste vekt, og 9 er ikke kjørbar.

Koden for A-Star-algoritmen ligger også klar til bruk i planleggingssystemets prosjektmappe, som også følger vedlagt på CD. Alle strategiene i planleggingsystemet kan benytte seg av denne koden, de må da inkludere filen $AStarUse.h$. Initialiseringen av rutenettet for verdensrepresentasjonen $InitMap$ settes fra initialiseringsdelen i *main* metoden. Det er valgt en størrelse på rutenettet på 30 x 30 centimeter, som gir en X på 10 og en Y på 7. Dette vil plassere blant annet batteriboksen i en rute, som må gis vekting ni, altså ikke kjørbar. Vektene kan settes ved hjelp av funksjonen $SetMapWeight(x, y, weight)$, fra alle funksjoner som inkluderer $AStarUse.h$. I starten av hvert spill blir det spurt om posisjonen til batteriboksen, denne posisjonen legges inn med vekting 9 i tilhørende rute.

4.3.5 Navigasjonsystem

Et foreløbig navigasjonssystem for roboten ble utviklet av Kristian M. Knausgård våren 2007. Her vil grensesnittet mot navigasjonssystemet presenteres. Først vil grensesnittet mellom navigasjonssystemet og planleggingssystemet bli presentert, deretter vil grensesnittet fra Proxy- / Gateway-systemet bli presentert. De to grensesnittene er vist grafisk i vedlegg H.2 og vedlegg H.3.
Det benyttes i meldingssendingen for roboten, en strukt for posisjon og en for waypoint. Strukten som benyttes for posisjon er $posisx\_robot\_state\_3dof$, og for waypoint benyttes $ai\_waypoint$

Grensesnitt mellom planleggingsystemet og navigasjonssystemet

`MINOR_NAVSYS_SET_POSITION`
Meldingen med dette MINOR nummeret sender en strukt med posisjonen til roboten. Denne blir brukt før spillet starter, og sender en melding til navigasjonssystemet om hvor roboten starter. Roboten starter enten i rødt hjørne eller i blått hjørne. Det er laget en posisjoneringsplate som settes i starthjørnet før start. Den setter roboten i samme posisjon hver gang. Posisjoneringsplaten har 45 graders vinkel, slik at startvinkelen alltid blir 45 grader sett fra startpunktet. X og Y punktet sett i forhold til hjørnet blir begge 250 millimeter.

`MINOR_NAVSYS_SET_WAYPOINT_ROBOT_RELATIVE`
Ved hjelp av denne meldingen kan man sette et nytt waypoint sett i forhold til midten av drivhjulakslingen til roboten.

`MINOR_NAVSYS_SET_WAYPOINT`
Her settes et globalt waypoint for roboten. Det vil si at det er et waypoint i spillebord koordinater.

`MINOR_NAVSYS_CLEAR_WAYPOINT`
Meldingen sletter alle waypoint som er sendt til navigasjonssystemet og venter på å bli behandlet.

`MINOR_NAVSYS_SET_SPEED_LIMIT`
Maksimal hastighet for roboten kan settes ved hjelp av strukten $speed\_limit$. Her kan både translasjonshastighet og rotasjonshastighet settes. Rotasjonshastighet er gitt i radianer per sekund, mens rotasjonshastighet er gitt i meter per sekund.

`MINOR_CMD_REQUEST_POSITION`
Kommandomelding, denne meldingen returnerer den faktiske posisjonen roboten befinner seg i. Strukten som returneres fra navigasjonssystemet er posisjonsstrukten $posix\_robot\_state\_3dof$.

`MINOR_NAVSYS_REACHED_WAYPOINT`
Hver gang roboten når et waypoint, vil den sende melding om dette. Da sendes strukten $ai\_waypoint$. Det kan sjekkes på posisjon, og identifikator som man sender med hvert waypoint, for å se hvilket waypoint som er nådd.

Grensesnitt mellom proxy-/gateway-systemet og navigasjonssystemet

`MINOR_CMD_SYSTEM_GO`
Dette er en melding til systemet at et spill er startet. Navigasjonssystemet skal nå gjøres klar til å ta imot startpunktet, og for deretter å ta imot waypoint roboten skal kjøre til.

`MINOR_CMD_SYSTEM_STOP`
Ved mottak av stoppmeldingen skal navigasjonssystemet skru av pådrag på motorene og vente på ny startmelding.

`MINOR_NAVSYS_SET_SPEED_LIMIT`
Samme melding som er beskrevet for grensesnittet mellom planleggingssystemet og navigasjonssystemet.

`MINOR_MOTOR_CONTROL_MESSAGE`
Her sendes en strukt til motorene med hastigheten man ønsker på de to motorene.

`MINOR_ODEMERTY_OBSERVATION`
Oppdatert informasjon fra kvadraturtellerene, som er tilkoblet løpehjulene på roboten, sendes i strukten *odemetry_observation*. Denne informasjonen inneholder den lengden hjulene har trillet fra forrige måling.

4.3.6 Bildebehandlingsystem

Bildebehandling er viktig for roboten, slik at planleggingsystemet får best mulig informasjon om miljøet den opererer i. I januar 2007 kom Lars Vråle inn i prosjektet og tok seg av robotens bildebehandlingen.

Det ble utviklet to bildebehandlingsprogram.

Et av programmene detekterer flasker, bokser og batterier samt posisjonen til batterisorteringsboksen, som har en av tre plasseringer på spillebordet (Eurobot Commitee, 2006).

Det andre programmet er tilknyttet et kamera som er montert inne i roboten. Dette programmet sjekker hvilken type objekt (flaske eller boks) flaske/-boks sortereren har plukket opp, og sender melding videre til planleggingsystemet om dette. Planleggingsystemet kan da eventuelt korrigere for om objektet flaske/boks sortereren har plukket opp er av samme type som roboten i utgangspunktet tror den har plukket opp, se avsnitt 4.3.4.

Bildebehandlingssystemene er satt opp til å sende meldinger til planleggingsystemet først når det kommer en gitt forespørsel. Planleggingsystemet kan spørre etter nærmeste posisjon på flasker, bokser eller batteri. Det kan

også spørre om plasseringen på batterisorteringboksen.

Det ble satt opp et grensesnitt mellom planleggingsystemet og de to bildebehandlingsystemene. Skjematisk diagram av grensesnittene er vedlagt i vedlegg H.1.

En strukt *cv_observation* som ligger i filen *observation_structs.h* sendes mellom bildebehandlingsystemet og planleggingsystemet. Denne inneholder en type-variabel som gir beskjed om hva type, flaske eller boks man enten spør etter eller bildebehandlingsystemet har observert. I strukten finnes det to variabler, x og y, for posisjonen av det observerte objektet. Denne posisjonen gis relativt til roboten, om det er observert flaske, boks eller batteri. Observasjonen av batterisorteringsboksen gir plasseringen av boksen i globale bordkoordinater. Batteriesorteringsboksforespørselen sendes bare en gang, og det er før roboten starter i sitt starthjørne. Lagets farge blir også sendt med til bildebehandlingsystemet. Lagfargen brukes til å bestemme plasseringen av batterisorteringsboksen, da denne får forskjellig relativ posisjon i forhold til roboten, alt etter hvilken side av bordet roboten starter på. Siste variabel i bildebehandligsstrukten er en variabel som gir vinkelen for eksempel en flaske eller boks ligger i, relativt til roboten.

4.3.7 Proxy- / Gateway-system

Det er utviklet et proxyprogram som kan sende meldinger til hele systemet, programmet fungerer også som en gateway mot CAN-Bussen. Programmet tar imot meldinger enten fra CAN-bussen eller fra posixkøer, og videresender disse til rett modul.

Ønskes en annen buss type enn CAN-buss for roboten, kan dette byttes ut i gateway-programmet, uten at resten av systemet påvirkes eller må skrives om.

Proxy-systemets gateway inneholder filen *can_bus.cpp* som tar seg av meldinger inn fra CAN-bussen, eller posix-køene. Initialiseringsfunksjonen for CAN-buss driveren ligger også i denne filen. Funksjonen
$can\_recieve\_message(TPCANMsg*, MessageHandler*, MessageHandler*)$
kjøres hver gang systemet får inn en ny melding fra CAN-bussen. Denne metoden tar inn pekere til to MessageHandler-objekt for meldingssending til henholdsvis navigasjonsystemet og planleggingsystemet. En peker til den mottatte CAN-meldingen tas også inn. En switch-case-metode inne i funksjonen velger rett handling for meldinger, basert på mottatt meldingsidentifikator, og sender meldingen videre til ønsket modul.

Proxy-systemet har egne filer for hver av de eksterne modulene, som kommuniserer med meldinger som inneholder data. For kommandoer, altså meldinger som ikke inneholder data, finnes alle metoder for å videresende disse i filen *proxyCommands.cpp*.

ProxyMotorControll.cpp
Filen *ProxyMotorControll.cpp* inneholder alle metoder som benyttes til styring av motorkontrollernodene samt kvadraturterene på disse 4.2.2. Det er og lagt opp til å kunne benytte kvadraturtellerene fra IO-kortet, uten annen endring i koden enn en define setning. For å benytte IO-kortets kvadraturtellere byttes identifikatoren til kvadraturtellersamplingen ut med IO-kortets kvadraturtelleridentifikator. En define i koden brukes til å velge kvadraturteller.
$\#define\ MOTORDRIVER\_QUAD\ 1$
gjør at modordriverens kvadraturteller benyttes, og ved
$\#define\ MOTORDRIVER\_QUAD\ 0$
benyttes IO-kortets kvadraturtellere. Define-settningen finnes i filen *main.h*. Samplingen av kvadraturtellerene kjøres fra main metoden hvert 20. ms, hvor mottatt kvadraturtellerdata sendes videre til navigasjonssystemet.

proxyCollisionDetection.cpp
Filen *proxyCollisionDetection.cpp* tar seg av all kommunikasjon med antikollisjonsystemet 4.2.4. her er det metoden
$proxySendCollisionDetected(\_\_u8*, MessageHandler*, MessageHandler*)$
som er den essensielle. Metoden tar imot meldinger fra antikollisjonsystemet hver gang det detekteres en kollisjon. Meldingen sendes da videre til planleggingsystemet som bestemmer hvilken handling som må utføres. Det er i denne metoden også lagt opp til å kunne sende melding direkte til navigasjonsystemet, slik at roboten kan stoppes uten innblanding fra planleggingsystemet. Planleggingsystemet er deretter ansvarlig for å starte roboten igjen.

proxyconfig.h
Oversikt over de brukte adressene på CAN-bussen finnes i filen *proxyconfig.h*. Gateway-systemet benytter denne filen for å transformere meldingsidentifikatorer fra CAN-buss til posix-meldingsidentifikatorer, som resten av programmene på hoveddatamaskinen benytter.

4.4 Mekanisk konstruksjon

4.4.1 Chassis

Chassiset på roboten er bygget etter tanken ved å ha to plan. Et plan der hoveddelen av robotens elektronikk befinner seg, og ett plan, det nederste, der sorteringsenheten for flasker og bokser er montert. Ved hjelp av denne konstruksjonen kan hele sorteringsenheten lett tas ut og byttes i andre enheter. Det er også mulig å sette inn en ny mekanisk enhet for senere års oppgaver, om det er nok plass for oppgaven. Plassen som er tilgjengelig for mekanisk enhet har en grunnflate på 822 kvadratcentimeter, og en høyde på 19 centimeter. Motorene benytter en del av dette arealet, men kan lett flyttes til andre posisjoner enn der de nå er montert. Dette volumet hadde vært tilstrekkelig for en stor del av oppgavene som er gitt tidligere år, og årets konstruksjon er bygget med tanke på videre gjenbruk.

Roboten ble designet i aluminium, og for å ha en omkrets på 117 centimeter. Reglene tilsier en omkrets før start på maksimalt 120 centimeter. To uker før avreise ble det montert sidedeksler. Det viste seg nå at roboten ble 121 cm i omkrets, og det måtte gjøres forandringer. Sidedekslene ble kortet inn og montert på toppen av bunnplata og ikke på siden, hvor de først var montert. Det ble også laget en ekstra knekk på sidedekslene slik at de fikk en innoverskråing bak. Undertegnede jobbet sammen med Hans Jørgen Berntsen på kybernetikkverkstedet for å oppnå omkrets som ville tilfredsstille Eurobotreglene. Sidedekslene er lakket med kromfarget lakk.

Frontplaten er også topplaten på roboten. Denne er festet i front med borrelås og på toppen med to tommelskruer. Frontplaten er sandblåst med to lakkerte striper bakover hele dekselet. Det er skjært ut hull til kollisjonssensorene og kamera i denne platen.

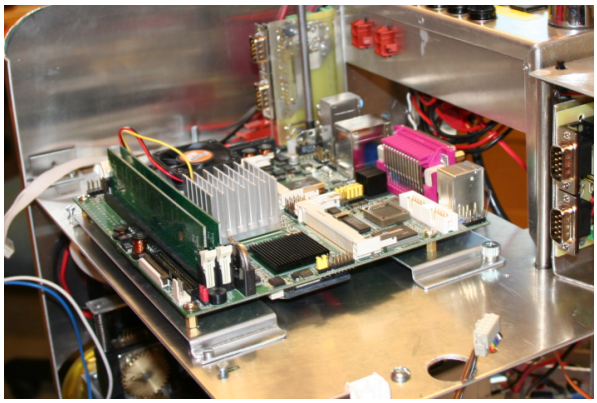
På toppen av roboten er det montert et tårn i gjennomsiktig plast. Tårnet er festet med borrelås.

Under skytemekanismens løftearmer i bunnplaten, ble det montert en plate for at ikke tannhjulene under armene skulle skrape nedi banen. Skraper roboten opp banen under en kamp vil laget bli diskvalifisert. Det er også to hull i bakkant av bunnplata for plassering av bakhjulene på roboten.

4.4.2 Hoveddatamaskin

Hoveddatamaskinen på roboten er montert på en aluminiumsplate. Denne platen er flat under hele hovedkortet. Utenfor hovedkortets kanter er platen bøyd ned slik at man har to kanter på en centimeter som tar nedi øvre planplate. I front av hoveddatamaskinens aluminiumsplate er det to ører som skyves inni to skruer. I bakkant er plata festet med to tommelskruer. Måten hoveddatamaskinen er festet på, gjør at hele datamaskinen kan fjernes fra roboten, ved hjelp av disse to tommelskruene. Det er skjært ut hull i aluminiumsplata for å kunne sette inn og ta ut Compact Flash kortet. Hoveddatamaskinen festet til

roboten er vist på figur 4.10. Over hoveddatamaskinen er det montert et deksel



Figur 4.10: Hoveddatamaskin

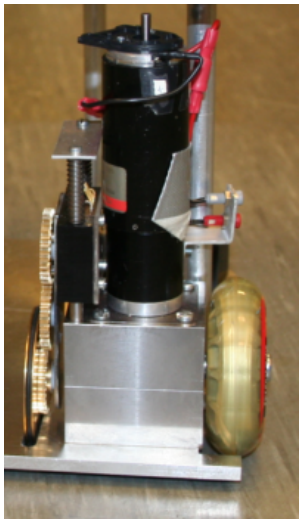
som har festeinnretning på samme måte som platen under hoveddatamaskinen. På dekselet monteres andre moduler.

4.4.3 Hjul og Motor-/gir-moduler

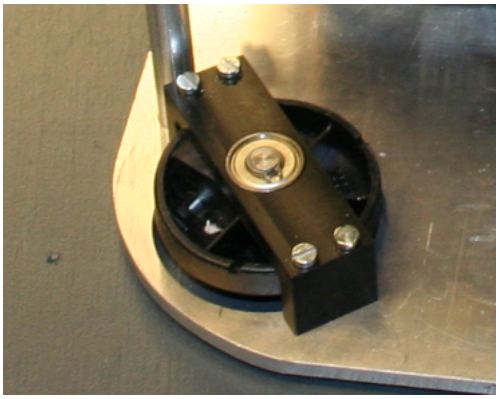
Under prosjektperioden ble det bestemt at motorene skulle stå i oppreist stilling i front på roboten. Det ble undersøkt om det var mulig å kjøpe ferdige girbokser med vinkeltannhjul for små elektriske motorer, noe som viste seg å være lite av på markedet. Etter å ha gjort en innsats for å kjøpe ferdige girbokser, ble det besluttet at disse måtte spesiallages til roboten. Kraftige vinkeltannhjul i stål ble bestilt fra Mekanex (Mekanex, 2006) i Sverige, og Hans Berntsen på kybernetikkverkstedet laget en kraftig girboks i stål til disse. Det ble benyttet rulleskøytehjul med diameter 76mm og hardhet 82A. Denne hardheten gir en god gripeevne for hjulene mot spillebordets overflate. Motorene ble deretter montert på toppen av girboksen. Girboksen kan kløyves på midten for inspeksjon av vinkeltannhjulene. Under drivmotorblokkene er det montert sentreringstapper, slik at motorblokkene ikke skal kunne skjevstiltes under bruk. Bilde av en av drivmotorblokkene er vist i figur 4.11. Før konstruksjonen av drivmotorblokkene var det ønskelig at også disse skulle være gjenbrukbare moduler, og ikke motorer festet rett i chassiset.

Løpehjul er viktig å ha på samme akse som drivhjulene. Det er lite tilgjengelig plass for disse i roboten, så det ble lagt ned stor innsats av kybernetikkverkstedet for å feste disse til motormodulen, og at de ble designet så smale som mulig. Løpehjulene sitter godt plassert på innsiden av motorblokkene med to fjærer for å gi tilstrekkelig press mot underlaget. Løpehjulene består av messinghjul med en gummi O-ring.

Bak på roboten er det montert to støttehjul. Dette er hjul fra støvsuger, som er festet med senteringtapp og kulelager på en brakett. Hjulene er avbildet i figur 4.12.



Figur 4.11: Girboks, motor, og løpehjul



Figur 4.12: Støttehjul

4.4.4 Antikollisjonssensorer

Det er montert tre antikollisjonssensorer, to av disse er montert på robotens øvre planplate. Den tredje sensoren er montert på midten av roboten ved siden av kamera. Det er skjært hull i frontplaten for sensorene.

4.4.5 Kamera

Lars Vråle som hadde ansvaret for bildebehandlingen i roboten monterte to kamera på roboten. Det ene er montert inne i roboten for å sjekke om det er flaske eller boks som er plukket opp. Det andre kameraet er montert på dekselet til hoveddatamaskinen midt over sorteringsenheten for flasker og bokser. Se figur 4.13.

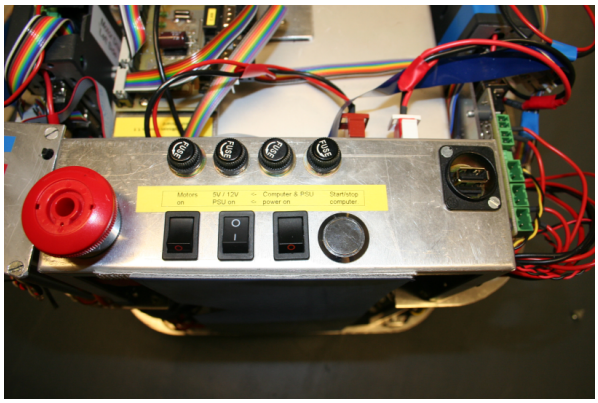
Figur 4.13: Kameraplassering

4.4.6 Koblingsmodul

Roboten er bygget mest mulig modulbassert. Det er da viktig å bygge en koblingsmodul for å knytte sammen de forskjellige delene på roboten. Koblingsmodulen er plassert på øvre plan helt bakerst på roboten. Modulen inneholder betjeningspanelet med brytere, indikatorlys og sikringer for roboten. Den interne spenningsforsyningen 4.2.3 er også montert på denne. Modulen er vist i figur 4.14. Modulen har to kontakter for spenningstilførsel til motordriverne i front. Alle ledningsgjennomganger uten kontakter har gummigjennomføringer se figur 4.15.

De tre bryterne som er montert på modulen sett fra venstre er som følger.
Bryteren lengst til venstre skrur av spenningsforsyningen til motorene.
Bryteren i midten skrur av og på det interne spenningsforsyningskortet.
Siste bryter benyttes til spenningstilførsel for hoveddatamaskin.

En trykkbryter produsert av Bulgin (Bulgin, 2007) er montert ved siden av spenningstilførselsbryteren for start av hoveddatamaskinen. Robotens nødstoppbryter er montert på venstresiden av koblingsmodulen.



Figur 4.14: Koblingsmodul

Figur 4.15: Gummigjennomføringer

Kapittel 5

Hoveddel Prosjektledelse

Ved å være prosjektleder for prosjektet må man holde rede på hva andre deltagere foretar seg, og at grensesnittene som er definert overholdes. Man må også være tilgjengelig for diskusjoner og hjelp for andre deltakere i prosjektet.

Kristian M. Knausgård og undertegnede hadde hovedansvaret for prosjektledelse og robotutviklingen sammen frem til og med desember 2006. I løpet av vårsemesteret 2007 har Kristian utviklet foreløbige systemer for navigasjonssystemet og antikollisjonsystemet. Kristian har hatt hoved kontakten med sponsoren og vært deltagende i diskusjoner om konstruksjonen av roboten samt innkjøp av deler.

Lars Vråle kom i januar inn i prosjektet, for å ta prosjekt på robotens bildebehandling, og ble da satt inn i grensesnittet for internkommunikasjon for roboten. Et grensesnitt for kommunikasjonen mellom bildebehandlingen og planleggingssystemet ble diskutert og skissert av undertegnede sammen med Lars. Bildebehandling er en viktig grunnstein for systemet, så det var viktig at Lars fikk jobbet med bildebehandlingen i et par måneder.

I mars begynte grunndelene av roboten som undertegnede hadde ansvaret for å komme på plass, så Lars kunne være med på flere av oppgavene. Før denne tid var der en god del påbegynte oppgaver som måtte ferdigstilles, før det var naturlig at Lars deltok i arbeidet med roboten.

Eksperter i Team-gruppen formulerte sin oppgave i januar. Deres oppgave for Eurobot var å designe og bygge sorteringsmodul for roboten. Undertegnede har etter beste evne prøvd å være tilgjengelig for gruppen gjennom hele vårsemesteret. Da elektronikken skulle styres ved hjelp av CAN-buss, fikk de utlevert en mikrokontroller AT90CAN128 og koden til CAN-bussen for denne. De ble også presentert et grensesnitt som var ønskelig for sorteringsmodulen, dette grensesnittet ble også utformet underveis i utviklingsfasen.

5.1 Organisering

5.1.1 Reise

Reisen til Frankrike ble bestilt tidlig i februar for å spare penger på økonomibilletter. Det ble bestilt billetter til masterstudentene, samt tre av eksperter i teamstudentene hadde mulighet til å bli med.

Før avreise ble det bekreftet på telefon og e-post at Eurobotorganisasjonen ordner med busstransport fra flyplassen i Paris til la Ferté-Bernard og tilbake. Da NTNU-laget var fremme på flyplassen, viste det seg etter at Eurobot organisasjonen var kontaktet, at bussen reiste en time før avtalt tid. Eneste løsning de kunne gi, var å ta toget til la Fertè-Bernard sammen med en Eurobot-representant som skulle ta tog fra Paris.

På hjemreisedagen ble det satt opp to busser til Paris sentrum. Der ble lagene selv ansvarlige for å finne videre reisemåte. Det ble brukt omtrent 300 euro mer enn planlagt på reisen på grunn av dette.

Innlosjering og mat ble dekket av Eurobotorganisasjonen for fire personer. For de to siste personene fra NTNU ble opphold dekket av sponsormidler.

Pakking av Robot

Under transport av roboten på fly er det svært viktig at roboten er forsvarlig pakket. Det finnes en reisekasse ved spillebordet som passer roboten perfekt. Denne benytter Eurobotlaget. Det er viktig at roboten pakkes slik at den står helt fast inni kassen ved hjelp av passende pakningsmateriale. Skrift med denne siden opp har ingenting for seg på boksen. På reisen til Frankrike ble robot-kassen observert liggende på siden på toppen av all annen bagasje på vei ut til flyet. På flyplassen i Paris ble en stuer observært etter innsjekking i det han tar fart og spenner i robotkassen, denne faller en halv meter ned fra transportbåndet og rett i et murgulv. Stueren så meget irritert ut da han oppdaget at kassen veide 35 kilo.

5.1.2 Sponsor

Kongsberg Gruppen ASA har for andre året på rad vært sponsor for NTNU Eurobotlaget, og har antydet at de ønsker et langsiktig samarbeid. Et langsiktig samarbeid vil frigi tid, som ellers ville blitt brukt på sponsorsøknader til robotrelatert arbeid. Eurobotlaget bestående av undertegnede, Kristian M. Knausgård og Lars Vråle, stilte 10. januar opp for sponsoren på deres bedriftpresentasjon på Studentersamfundet. Her ble fjordårets og årets robot vist frem for de fremmøtte på en stand. Eurobot deltagerne startet en uke før semesterstart for å gjøre forberedelser til denne presentasjonen.

Eurobot laget har vært fornøyd med Kongsberg Gruppen ASA som sponsor. Lag-t-skjorter med Kongsberglogo ble mottatt fra Kongsberg Gruppen ASA en uke før avreise, og Lars Vråle tok seg av videre trykking av disse. Det ble

også mottatt klistremerker for roboten, og diverse reklamemateriell som ble tatt med til Frankrike.

5.1.3 Budsjett og konto

Eurobot prosjektet trenger mer økonomiske midler enn Instituttet kan stille med. Det har hvert år vært nødvendig med sponsorstøtte. Det har for tidligere år vært vanskelig å følge med hvor mye av budsjettet som er gjenværende da Eurobot ikke har hatt egen konto. Denne problematikken oppstår blant annet da avregninger fra komponentlageret og verkstedet ikke kommer når deler er tatt ut, men på vilkårlige tidspungt gjennom semesteret. Når varer bestilles fra utlandet, kan toll og moms belastes i ettertid av regningen fra leverandørene.

Både instituttets administrasjon og Eurobotdeltagerne har hatt problemer med å få oversikt over alle transaksjoner som belastes Eurobot. Da Eurobot hovedsakelig drives av sponsormidler, har det i år blitt opprettet eget prosjektnummer og konto for Eurobot, slik at sponsormidler ikke blandes med instituttets penger. Etter at denne kontoen er kommet på plass, vil neste års Eurobotlag vite ved høstsemesterets start hvor mye midler de har til rådighet før de søker om sponsormidler. Som prosjektleder har også undertegnede kunnet ha en god oversikt over hvor mye av budsjettet som til en hver tid er brukt.

Det viste seg å være vanskelig å finne ut hvor mye av budsjettet fra Eurobot 2006 som ikke var brukt. Det ble da overført 15000 kroner som et startbeløp til årets lag, da dette var omtrent det som gjensto fra fjordåret. Sponsorstøtten fra Kongsberg var på 65000 kroner. Det ble også innvilget 15000 kroner i støtte fra fakultetet. Disse pengene skulle ikke overføres til konoen uten at de ble brukt.

5.1.4 Eurobot-forenings medlemskap

Før avreise ble det innbetalt 50 Euro til Eurobotkomiteen. Dette er betaling av medlemskap i Eurobotforeningen, og gjelder for hele laget frem til August 2007. For utenlandsbetaling er det viktig at man har både IBAN- og SWIFT-nummer. IBAN er kontonummer, og SWIFT er identifikasjonen av banken som pengene skal overføres til. Eurobotorganisasjonen hadde bare oppgitt IBAN-nummeret, og av den grunn ble det forsinkelser på innbetalingen av pengene. Det anbefales for senere års Eurobotlag å ta med penger for medlemskapet, og betale dette ved ankomst til konkurransen.

5.2 Utførsel av Robot og utstyr til EU land

I Januar kontaktet undertegnede tollvesenet for å undersøke regler for utførsel av roboten og diverse utstyr til utlandet. Det ble da opplyst at man trenger et ATA Carnet skjema, dette skjema muliggjør midlertidig innførsel av varer til 63 land. Dette er dokumentasjon på at man skal ta utstyr man har med

seg inn i et land ut igjen. Ved hjelp av ATA Carnet skjema vil dette ikke være noe problem. Med tanke på at roboten og annet verdifullt utstyr man trenger under konkurransen i Frankrike skal kunne taes med tilbake til Norge uten å måtte betale toll på dette, brukes også ATA-carnet skjema for dette. ATA Carnet skjema kan bestilles hos Midt-Norsk Handelskammer i Trondheim (Midt-Norsk Handelskammer, 2007).

Et ATA-CArnet skjema ville koste rundt 3000 kroner. 4. mai kontaktet undertegnede igjen tollvesenet, denne gang Værnes tollkontor. Etter å ha pratet med Tollbetjent Haga ble det enighet om at ATA-Carnet ikke var nødvendig. Det som ble avtalt at skulle gjøres, var å møte opp hos tollvesenet der man får et skriv på hva som ble brakt ut av landet. Dette skal deretter vises når roboten blir brakt tilbake til landet.

5.3 Utførsel av varer til reperasjon

Eurobot har ved flere anledninger sendt varer til reparasjon i utlandet. For å unngå å betale toll og moms på varen når den returneres til Norge, er det viktig at en proformafaktura fylles ut. Fem av MaxBotix ultralydsensorene som ble kjøpt inn i januar måtte returneres på denne måten. Proformafakturaen som ble benyttet finnes i vedlegg G.1.

Kapittel 6

Resultater og Diskusjon

6.1 Eurobot Open 2007

Eurobot konkurransen fant sted torsdag 17. mai til søndag 20. mai 2007 i La Fertè-Bernard. Konkurransen ble arrangert i et messe- og idretts-område. Alle lagene i Eurobotkonkurransen får tildelt en arbeidsplass som de kan benytte under hele konkurransen.

Parallelt med Eurobot Finalen ble også den franske Eurobot konkurransen arrangert på samme sted. På grunn av at den franske og den internasjonale konkurransen ble arrangert samtidig, ble det ikke noe skille mellom de forskjellige konkurransene når det gjaldt arbeidsplasser. Dette førte til at arbeidsplassområdene ble fullstendig dominert av franske lag, og det kunne virke som om den internasjonale konkurransen var nedprioritert. Dominansen av franske lag kommer av at der deltar nesten 200 lag fra Frankrike, og der var ikke mer enn 50 internasjonale lag.

Disse observasjonene kommer på bakgrunn av undertegnedes deltagelse i Eurobot Open 2006 i Italia. Der var den italienske finalen ferdig den dagen de internasjonale lagene ankom. Her var det omtrent 60 internasjonale lag totalt og alle hadde arbeidsplass i samme hall. Dette gjorde at det var en internasjonal ramme rundt hele arrangementet. Denne rammen manglet i Frankrike.

6.1.1 Konkurransen

Konkurransen foregår ved at roboten først må kvalifiseres for deltagelse i Eurobot. Kvalifiseringen består av tre tester.
Den første testen går på å tilfredsstille reglene for mekanisk konstruksjon av roboten.
For å bestå den andre testen må roboten vinne et spill, uten andre roboter på banen.
Den siste testen som tas er å teste at antikollisjonsystem et fungerer.

Det er en begrensning på fem forsøk for å bestå de to siste testene. Man må vinne et spill iløpet av disse fem forsøkene, og antikollisjonsystemet må fungere. Når roboten er kvalifisert for deltagelse, er man sikret å spille fem spill. Etter at alle lagene har spilt fem spill, vil de 16 lagene med mest poeng gå videre til finalerundene. Deretter blir det vinnerne som hele tiden går videre, til to lag står tilbake i en finale.

For roboten fra NTNU forløp konkurransen seg som følger:

Alle kravene til mekanisk konstruksjonen av roboten var oppfylt, så denne testen ble raskt bestått.

Neste test i kvalifiseringen besto av å vinne et spill uten motstander. Her startet roboten på blå side. Det var i planleggingsystemet lagt inn at den skulle kjøre til andre siden av bordet, for deretter å benytte kamera for å se flasker og bokser. Roboten plukket opp første boks den prøvde seg på, og kjørte til riktig sorteringsbeholder, og skjøt boksen oppi. Deretter snudde den seg rundt og prøvde seg på to flasker til, men bommet på disse.

Siste test var å se at antikollisjonsystemet virket. Her ble det satt en etterligning av en robot på banen, og roboten skulle kjøre mot denne og stoppe. Roboten stoppet på omtrent 25 centimeter fra robotetterligningen.

Kvalifiseringen av roboten ble altså gjennomført på første forsøk, og det var nå klart for å spille de fem obligatoriske kampene.

I første kamp møtte NTNU-roboten motstanderens robot omtrent midt på banen. Her ble de to robotene sittende fast i hverandre, og kampen utartet seg ikke til noe mer. Kampen endte uavgjort.

Kamp to ble ikke mer spennende enn den første da robotene også stoppet mot hverandre rundt midten av banen. Denne kampen endte også uavgjort.

Før kamp nummer tre omprogrammerte undertegnede *motorAi*-klassen i planleggingsystemet. Med en gang roboten oppdaget en kollisjon skulle den utføre en unnamanøver. Ble det detektert en kollisjon på venstre sensor, skulle den kjøre et waypoint 30 cm ut til høyre side og motsatt for sensorene på høyre side. Detekterte sensoren i midten en kollisjon ,skulle roboten snu og kjøre 50 centimeter i 45 grader bakover. Etter disse unnamanøvrene skulle roboten gjenoppta strategien som var aktiv. Om det kom deteksjon på flere sensorer samtidig, ville bare en av handlingene utføres. Når kampen ble satt igang møte roboten den andre roboten på omtrent samme måte som den hadde gjort i de to første kampene. Omtrent 30 centimeter fra den andre roboten kjørte roboten rett inn mot midten av bordet, for så å fortsette fremover. Den andre roboten ble fint passert, og roboten kjørte til motsatt side av bordet fra startpunktet. Da den skulle begynne å plukke flasker og bokser, stoppet roboten og gjorde ikke mer. Den andre roboten fikk heller ikke sortert flasker eller bokser i denne omgangen, så kampen endte også her uavgjort.

Mellom kamp tre og fire var det ikke mye tid til programmering. Undertegnede brukte denne tiden til å finne ut hvorfor roboten stoppet i tredje kamp. Det ble ikke funnet noen opplagt feil, og da roboten ble testet, oppsto ikke

feilen igjen. Fjerde kamp ble startet med omtrent det samme programmet som tredje kamp. Roboten møtte også denne gangen den andre roboten på midten av banen. Nå snudde roboten og kjørte en halv meter bakover, deretter ble den stående å kjøre frem og tilbake innen en meters radius. Da antikollisjonsystemet er implementert med lysdioder som blinker når det detekterer kollisjon, ble det observert at sensorene detekterte kollisjoner hele tiden uten at noe sto foran roboten. Roboten sto da slik hele kampen, til den stoppet på tid.

Før siste obligatoriske kamp ble det undersøkt hva som kunne forårsake at antikollisjonsystemet slo inn uten at noe sto foran roboten. Det ble ikke funnet noen annen forklaring enn at både ultralydsensorene og de infrarøde sensorene ble utsatt for interferens av andre roboter. Inne i hallen der konkurransen foregår er der tre spill som går samtidig på tre bord. Det er mulig at antikollisjonsystemet ble forstyrret av andre roboters ultralyd og IR-sendere, da de fleste roboter har denne typen antikollisjonsystem. Utenfor spillehallen detekterer roboten ikke kollisjon uten at der faktisk er kollisjon.

Da de fleste robotene, og alle som NTNU roboten hadde spilt mot i de fire første kampene kjørte direkte mot motsatt side av bordet etter start, var det stor sannsynlighet for at dette ville skje også i femte kamp. Det ble bestemt at roboten skulle ha innkoblet antikollisjonsystemet til den hadde kjørt forbi den andre roboten, for deretter å slå av dette. Sett ut fra de fire første kampene var det sannsynlig at roboten ville ha kommet forbi den andre innen 15 sekunder, så det ble lagt inn en timer i proxy-/gateway-systemet som gjorde at etter 15 sekunder ble ikke lenger kollisjons meldinger sendt videre til planleggingssystemet. Da spillet var igang, forløp det seg som i de fire første kampene, roboten møtte den andre på midten av banen, tok en unnamanøver og fortsatte til motsatt ende av spillebordet fra der den startet. Roboten snudde seg fint mot flaskene og boksene og kjørte mot den nærmeste boksen og plukket opp denne. Boksen ble liggende helt på enden av løftearmene, da et batteri fra banen hadde kilt seg inn bak bak disse armene. Roboten kjørte da mot sorteringskurven for bokser, og ble stående i perfekt posisjon før den prøvde å skyte. Boksen falt rett ned når det ble skutt, da den ikke var kommet langt nok bak på løftearmene. Deretter dyttet ryddearmen, som var satt inn under løftearmene, batteriet bort og armene ble senket ned igjen. Roboten var nå klar til å se etter nye flasker og bokser, men kjørte deretter rett i kanten av bordet og ble stående fast. Den mest sannsynlige årsaken til at roboten kjørte i kanten, var nok at batteriet hadde løftet ene løpehjulet opp fra bakken slik at navigasjonssystemet trodde roboten sto i en annen vinkel enn den gjorde, og kjørte utover istedenfor innover på banen. Også denne kampen endte uavgjort.

6.1.2 Diverse

Som beskrevet i innledningen kom NTNU laget på 25 plass i turneringen, der det deltok 39 lag. Det møtte 50 lag, men 11 lag klarte ikke å gjennomføre kvalifiseringen. Flere lag enn 50 var i utgangspunktet også påmeldt, men møtte ikke opp til konkurransen.

Under årets konkurranseperiode ble de andre lagenes roboter studert. Ikke mer enn omtrent fem roboter viste seg å være nevneverdig bedre en de resterende. Disse robotene endte også på toppen av resultatlisten. Lagene fikk også betydelig flere poeng enn de resterende lagene. De beste robotene klarte å lagre flere bokser og flasker i roboten samtidig, og selve sorteringen ble blandt de fleste av disse gjort ved å skyte objektene til riktig beholder. Disse robotene bar preg av også å ha gode strategiplanleggingsystemer.

De aller fleste deltagende roboter kjørte derimot faste ruter på bordet der de visste hvor flasker og bokser eller batteriene sto i utgangspunktet. Det ble også observert at flere lag kjørte rundt faste ruter til de fikk et objekt å sortere inn i roboten. Mange av lagene hadde bygget så enkle roboter som mulig og gikk for strategien å kjøre på flasker og bokser, slik at fra en til tre spratt opp i en sorteringsbeholder. Minuspoeng får man ikke før etter tredje feilsorterte flaske eller boks. Ved få to flasker og en boks oppi beholderen for bokser, vil man få 1 poeng selv om man har to feilsorterte. Lagene som hadde valgt en slik strategi, fikk da fire poeng når det andre laget ikke klarte sorteringsoppgaven. Få av lagene hadde gode strategiplanleggingsystemer og sensorer som kamera til å detektere flasker og bokser, før de kom inn i roboten.

To uker før konkurransen sendte Eurobotkomiteen ut informasjon til de påmeldte lag (Eurobot Commitee, 2007). På grunn av at flere roboter i de nasjonale finalene hadde hatt problemer med kvalifiseringen, oppfordret komiteen til å holde strategiene enkle, og lage egne kvalifiseringsprogrammer.

6.2 Grunnleggende Systemer på Roboten

6.2.1 Innebygget Hoveddatamaskin

Hoveddatamaskinen på roboten Commell LV-677 som har vært brukt på roboten dette året, har som beskrevet i kapittel 4.2.1 medført en del uforutsett ekstraarbeid. De fleste problemene kan knyttes til BIOS. Med BIOS versjon 1.4 har det derimot ikke vært observert problemer. Eurobotlaget har gått til anskaffelse av to like hovedkort, for å kunne ha ett i reserve. Det første hovedkortet som var installert på på roboten defekt under testperioden. Her var det nettverkskortet ikke lenger responderte.

En avgjort fordel med denne hoveddatamaskinen er hovedkortet som kan kobles direkte til batterier uten ekstern spenningsregulator. Hoveddatamaskinen trekker mellom 1.5 og 2 Ampere. Dette må man ta med i beregningene på hvor lenge man ønsker å kjøre maskinen før batteriet må byttes.

Formfaktoren på hovedkortet er MINI-ITX, som har en størrelse på 17x17 centimeter. Denne har vært ideell for robotens konstruksjon.

Hoveddatamaskinen har fungert fint siden bytte av hovedkort, og videre bruk av denne er å anbefale. Skulle man derimot ønske å bytte ut denne maskinen på grunn av de tidligere feil på BIOS og nettverkskort, kan en ultraportabel bærbar pc være et godt valg. Denne vil få plass der hoveddatamaskinen nå er plassert. Fordelen med en slik pc er i hovedsak tastatur og skjerm som kan være greit å ha under test av roboten. Man vil også under kampene kunne starte roboten direkte fra denne pc-en og ikke være avhengig av å starte roboten gjennom SSH fra en annen pc. Batterilevetiden på en slik pc kan også være avgjørende, da disse kan ha flere timers arbeidstid, og batteriet er innebygget i pc-en. Det er viktig å velge en pc med IDE-harddisk, slik at denne kan byttes ut med et CompactFlash kort.

6.2.2 Motor Pådragssystem

Flere versjoner av motordriver-modulen er utviklet, og man har endt opp med en modul som har vist seg å tilfredsstille ønskede krav til funksjonalitet og driftssikkerhet. Etter flere varianter av modulen er siste variant en driver versjon 3.0. Det har vært lagt mye arbeid i å utvikle en robust motordriver både når det gjelder programvare og hardware. De oppnådde målene med å designe en motordriver-modul er som følger:

- Styrt med CAN-buss.
- Plattformuavhengig, er gitt av bruken av CAN-Buss.
- Motorene kan hastighet- eller moment-styres.
- Mulighet for å lese av fra CAN-Bussen faktisk hastighet eller momentet motoren utvikler.

- Distribuert system, da regulatorer lar seg implementere på noden. Man kan også implementere regulatorer på fra andre CAN-Buss-noder.

Motordrivernode versjon 3.0 har vist seg å fungere godt for roboten. Den tar liten plass og kan hastighetsregulere eller momentregulere motorer. Man kan også lese av moment og hastighet denne utvikler fra CAN-bussen. Modulen anbefales fortsatt brukt på senere års roboter.

6.2.3 Intern Spenningsforsyning

Det har vært benyttet to spenningsforsyninger på roboten.

Logikk-spenningstilførselen til CAN-nodene har vært hentet fra den svitsjede spenningsforsyningen. Dette har vært en klar fordel da spenningstilførselen har benyttet samme kabel og kontakter som CAN-buss-signalet. Svitsjet spenningsforsyning har også vært en fordel med tanke på det lave effekttapet disse har. Da roboten kjører på batteri, ville et høyere effekttap ført til kortere kjøretid for roboten.

Den lineære spenningsforsyningen har vært benyttet til robotens servoer. Servoer inneholder motorer som kan gi støy på spenningstilførselen når de belastes hardt, noe som ikke ville vært heldig for logikkretser.

Det er ikke observert noen problemer med bruk av disse spenningsforyningene.

6.2.4 Antikolisjon

Antikollisjonsystemet har fungert fint under testperioden, men var derimot medvirkende årsak til problemer i noen av kampene under konkurransen. Systemet har lysdioder som viser når en kollisjon inntreffer. Det ble under noen av kampene observert at antikollisjonsystemet detekterte kollisjoner når ingen ting sto foran roboten. Dette skjedde bare på konkurransebordene og i hallen der konkurransen foregikk.

Årsaken kan være andre roboter som benyttet samme antikollisjonssystem med ultralyd eller infrarødt lys. Disse ville da kunne påvirket antikollisjonssystemet på NTNU-roboten. En annen årsak kunne vært flere refleksbånd som var plassert rundt på banen. Disse kunne ha reflektert signalene mer effektivt enn et objekt uten reflekser, slik at det oppsto falske målinger.

Roboten som vant konkurransen RCVA, fra det Franske universitetet Paris X, benyttet laser for å detektere den andre roboten. Den plasserte da en modul på tårnet til den andre roboten som ble detektert av laserstrålen de selv hadde i tårnet. Dette så ut til å fungere utmerket.

Det anbefales å teste antikollisjonsystemet nøye før dette brukes igjen. Systemet må testes sammen med reflekser og støysendere for infrarødt lys og ultralyd.

6.2.5 IO Kort

Kortet har i hovedsak blitt benyttet til reint IO kort. Brytere for valg av lagfarge og omstart av spill er implementert, samt at startsnoren også har vært tilkoblet på dette kortet. Alle disse signalene blir sendt direkte videre til planleggingssystemet når de bli aktivert.

Kortet har implementert to kvadraturtellere som har vært ment som reserve for de som er implementert på motordrivermodulene. Kode for disse er implementert og kan tas i bruk ved behov. Kvadraturtellerene ble ikke benyttet på roboten, da det ikke oppsto problemer med kvadraturtellerene på motordrivermodulen.

Batteri-sortereren var også ment å koble til dette kortet, som har flere servoutganger, digitale innganger og utganger ledig. Da batteri-sortereren ikke ble ferdigstilt, ble det ikke behov for disse utgangene.

Kortet kan gjenbrukes til en rekke funksjoner på en fremtidig robot, da det alltid vil oppstå situasjoner der man trenger en ledig IO-pinne. Fargevalg og startsnor vil det alltid være behov for.

6.2.6 Flaske-/Boks-sorterer

Modulen som ble utviklet av Eksperter i Team-gruppen hadde en del problemer da den ble montert i roboten. For å løse oppgaven var roboten helt avhengig av at denne fungerte prikkfritt. Flere forbedringer ble gjort på systemet som beskrevet i kapittel 4.2.6. Etter at disse forbedringene var utført, fungerte systemet fint.

Flaske-/Boks-sortereren er riktignok sårbar med tanke på de mekansiske armene som stikker ut av roboten. Det hadde vært en fordel om alt var samlet inne i roboten. Da man både skal plukke opp flasker og boker, for så å skyte disse, er dette derimot ikke noen enkel oppgave, da dette nødvendigvis må ta en del plass.

Under selve konkurransen ble det ikke observert noen problemer med sorteringsmodulen.

6.2.7 Batteri Sorterer

Grunnet flere uforutsette hendelser i ukene før avreise til Frankrike, ble batterisortereren ikke ferdigstilt. Dette førte til at verken strategien for planleggingsystemet eller koden for batterisortering på IO-kortet aldri ble testet på roboten. Det som står igjen for at systemet skulle være ferdigstilt, er frontplata som skal monteres foran på tannstangen, ikke er produsert.

Modifiseringen av servoene som ble foretatt fungerte fint, og tilsvarende modifisering kan anbefales for tilsvarende oppgaver i fremtiden.

6.2.8 Servoer

Servoene som er benyttet i roboten er solide servoer med metallgir. Servoene er raske og sterke. Ingen problemer er observert med noen av servoene.

6.2.9 Batterier

Bruken av LiPo-batterier har fungert tilfredsstillende. Disse batteriene har stor kapasitet og tar liten plass i roboten, og har lav vekt. Batteriet som benyttes på hovedkortet har en kapasitet på 2200mAh, noe som gir en kjøretid på roboten like under en time. En time er i minste laget om man skal spille to kamper like etter hverandre, og da det ofte er ventetid før kampene. Under ventetiden er robotens hovedkort startet og alle programmer er klare til å kjøres. Ved feilsøking og testing av roboten er også en time kort kjøretid.

Undertegnede anbefaler bytte av dette batteriet til et som gir minimum to timers kjøretid, og da et batteri på minimum 4500 mAh.

6.2.10 Kontakter

Ved å benytte kontakter og ikke skrutilkoblinger for alle ledninger og moduler har vært en klar fordel. Dette har ført til at montering og demontering av hele roboten kan utføres raskt. Ved å benytte forskjellige typer kontakter på de forskjellige moduler og forskjellige spenninger, har også vært en fordel da sannsynligheten for feilkobling minimeres. Forskjellige farger har også vært benyttet på kontaktene på flere moduler, for å lettere se hvor tilhørende kontakt skal kobles.

Å benytte kontakter på alle moduler anbefales til alle moduler som senere vil bli bygget til robotene.

6.3 Programvare

6.3.1 Operativsystemet

Operativsystemet Linux Ubuntu Server har vært et godt valg av operativsystem på roboten. Det er ikke observert feil på roboten som direkte skyldes operativsystemet. Linux-kjernen som er installert på roboten har versjonsnummer 2.6.15. Det har vært viktig å ikke oppdatere kjernen på Ubuntu distribusjonen da dette ville medført at drivere som er installert, også måtte oppdateres. Grafisk brukergrensesnitt er ikke implementert på serverversjonen av Ubuntu, noe som gir plassbesparing på harddisken.

Pakkebehandlingsprogrammet Advanced Packaging Tool «APT» er standard på Ubuntu. Ved hjelp av APT kan programmer på en rask og enkel måte installeres, avistaleres og oppdateres. APT har vært en avgjort fordel å ha installert da man sparer mye tid på innstalasjon av nye programmer.

Secure shell «SSH» har vært benyttet på roboten, for ikke å være avhengig av å ha tastatur og monitor tilkoblet. Det har også vært en fordel å benytte SSH, slik at flere brukere har kunnet jobbe på roboten samtidig.

Undertegnede anbefaler senere års Eurbotlag å fortsette bruken av Ubuntu Linux som operativsystem på roboten.

6.3.2 CAN-Buss

CAN-Bussen som er valgt til komunikasjonsbuss på roboten har fungert utmerket, og har vært bærebjelken for et modulbasert system. Ved å benytte CAN-bussen sammen med posix-meldinskøsystemet på hoveddatamaskinen, er all kontroll av roboten meldingsbassert, noe som har gitt klare definerte grensesnitt for hver modul.

CAN-buss-systemet har forenklet samarbeidet mellom Eurobotlagets deltagere, da deltagerne bare må forholde seg til grensesnittet mellom modulene. All kommunikasjon mellom de forskjellige systemene har gått på meldingssending, som igjen har gjort det lett å finne hvilke delsystemer som inneholder feil når roboten ikke har utført tiltenkt handling.

Når moduler designes med CAN-buss, trenger utvikleren bare å forholde seg til grensesnittet, og ikke hele systemet. Simulering er også en enkel oppgave, ved hjelp av et CAN-buss terminalprogram kan meldinger sendes til og fra modulen.

Under konkurranseperioden i Frankrike ble de andre deltagerlagenes roboter studert. Flere lag hadde implementert CAN-buss på roboten, og hadde samme positive erfaringer som undertegnede.

Det anbefales å benytte CAN-buss videre for Eurobot robotene, da denne har vist seg robust, og gir et oversiktlig grensesnitt mot de forskjellige moduler. Det finnes nå flere veldokumenterte CAN-buss-moduler tilgjengelig for bruk på senere års roboter.

6.3.3 Posix-kø-systemet

Posix-kø-systemet har sammen med CAN-bussen gjort hele roboten modulbasert. Systemet har gjort programmene som kjører på robotens hoveddatamaskin modulbaserte, med klare definerte grensesnitt.

Under utviklingen av programmer som skal benytte posix-kø-systemet har man kunnet simulere oppførsel ved å sende meldinger inn på systemet, samtidig som meldingene som går ut av systemet logges. Dette har vært en avgjort fordel for å teste programmer, før programmene som det skal kommuniseres mot er ferdigstilte.

6.3.4 Strategiplanleggingssystemet

Det har tidligere vært gjort lite arbeid når det gjelder planleggingsystemer for Eurobotrobotene til NTNU. I januar ble det besluttet å utvikle et planleggingsystem som skulle benyttes på årets robot. Rammeverk for planleggingssystemet som baserer seg på designpatteren Observer, ble implementert. For å kunne benytte planleggingssystemet anbefales det å sette seg inn i designpatteren Observer (Gamma, E. ,Helm, R. , Johnson, R. ,Vlissides, J., 1995).

Krav til planleggingsstemet

Det ble satt en del krav til rammeverket i kapittel 4.3.4. De forskjellige kravene og oppnåelsegraden vil her bli diskutert.

- `Ha mulighet for et stort antall forskjellige uavhengige strategier.`
  Kravet er oppnådd da det ikke er satt noen øvre grense for hvor mange strategier man kan ha. Hver strategi implementeres som et eget objekt. Ved å implementere strategiene som egne objekter, vil heller ikke en feil i en strategi gi følgefeil til en annen strategi.

- `Proritere målorienterte strategier.`
  Kravet oppnås ved måten en strategi implementeres på. Rammeverket bryr seg ikke om oppførselen til strategiene, eller hva mål de har. Skal systemet prioritere målorienterte strategier, må strategien selv returnere en høy prioritetsverdi, om den ser at den kan nå et mål.

- `Velge strategi ut fra hvilken strategi som har høyest prioritet.`
  Kravet løses i metoden *GetAgentPriority*() i objektet *PlansysObservation*. Denne metoden inneholder en algoritme som velger en ny strategi, så lenge ingen andre er aktive. Strategien som blir valgt er den strategien som gir høyest prioritet av alle implementerte strategier.

- `Hver strategi skal ha mulighet til å ha all tilgjengelig oppdatert informasjon til en hver tid.`
  Kravet var en av de viktigste kriteriene til planleggingsystemet, og hovedgrunnen til at designpatteren Observer ble valgt. Hver gang en ny melding kommer inn til systemet, blir alle strategier underrettet om hvilken type melding som er kommet. Strategiene bestemmer da selv om de skal hente meldingen inn til strategiene eller ikke.

- `Robust ved at den kan fjerne strategier som ikke virker.`
  Muligheten for å fjerne strategier som ikke virker kan gjøres på flere måter. Variabelen *agentPriority* i strategien kan settes negativ, eller returnere negative verdi fra metoden *GetAgentPriority*, da vil ikke strategien settes aktiv. Hele strategien kan også fjernes fra Vektor-listen ved hjelp av metoden *Detach* i Subject klassen, som *PlansysObservation*-objektet er arvet fra. Finner man ut at det er en implementert strategi som ikke virker før man kompiler programmet, kan man enkelt fjerne strategien, ved å markere ut opprettelsen av objektet.

- `Må kunne håndtere avbrudd uten at dette påvirker aktive strategier.`
  Ved hjelp av metoden *GetInterruptAction* i strategiene, og strukten *interrupt_controll_struct*, kan kommandomeldinger og waypoint settes fra alle strategier uten at dette påvirker aktiv strategi. Waypoint som settes i *interrupt_controll_struct* blir sendt til navigasjonssystemet som avbrudds-waypoint. Disse waypoint vil bli kjørt med en gang de ankommer navigasjonssystemet, selv om navigasjonssystemet har flere waypoint som venter på å bli kjørt.

- `Aktiv strategi skal beholdes til den har oppnådd sitt mål, eller en annen strategi gir betydelig høyere prioritet.`
  Metoden *GetAgentPriority*() i objektet *PlansysObservation* tar seg av å beholde en strategi til denne er utført. En strategi kan derimot sette flagget *clearActiveStrategy* i *interrupt_controll_struct* til *true*. Settes dette flagget *true*, vil aktiv strategi avbrytes og strategien som har satt *clearActiveStrategy*-flagget til *true*, kan sette så høy prioritet på neste prioritersforespørsel at den mest sannsynlig blir valgt.

Diverse

Strategiene i planleggingssystemet kan gjerne være tilstandsmaskiner, og man kan ha flere strategier for å gjøre det samme, som da å sortere flasker eller bokser. Ser man at en av strategiene viser seg å fungere bedre enn en annen, fjerner man raskt denne strategien ved å ikke opprette objektet for den i initialiseringen.

Fordelen med å implementere planleggingssystemet som det her er gjort, er om man ønsker en ny strategi, kan denne implementeres uten å forandre rammeverket, eller eksisterende strategier. Hadde planleggingsystemet vært implementert som en stor tilstandsmaskin, ville en hver forandring kunnet få konsekvenser for resten av tilstandsmaskinen.

Implementert planleggingsystem

Det implementerte rammeverket for planleggingsystemet har under konkurransen i Frankrike og under testing av roboten, vist seg å fungere tilfredsstillende. Ved hjelp av designpatteren Observer oppdateres alle strategier i systemet når en ny melding ankommer. For å holde kontroll på når strategier blir oppdatert leses meldingene ut av posix-køene uten avbruddshåndtering som det er mulighet for. På denne måten, vil alle de nyeste meldingene planleggingsystemet har fått inn være oppdatert, før man begynner å lese ut prioriteter eller utfører en handling på en strategi. Ved å lese ut meldingen på denne måten vil man slippe å benytte semaforer eller mutex-låser når man oppdaterer strategiene.

Kommunikasjonen mellom planleggingsystemet benytter posix-kø-systemet, men er ikke låst til å benytte dette. Et hvilket som helst meldingssystem kan benyttes. Hele grensesnittet mot resten av roboten befinner seg i filen *mainplansys.cpp*. Å legge grensesnittet i denne filen gjør at planleggingsystemet kan benyttes til helt andre oppgaver enn styring av roboten.

På det implementerte planlegginsystemet er det benyttet tre av de fire strategier, hvor ene strategien *MotorAi* er en ren avbruddsstrategi. De to andre strategiene som er i bruk er *BottleBoxSort* og *FirstStrategy*, som begge bare bruker *control_struct* til kommandoer, og gir ingen meldinger til avbruddsstrukten *interrupt_control_struct*.

FirstStrategy

Strategien *FirstStrategy* settes aktiv når planleggingssystemet initialiseres. Denne kjører roboten en rute for å komme til andre siden av bordet hvor fra starter.

BottleBoxSort

Etter at *FirstStrategy* har gjort seg ferdig, er det bare *BottleBoxSort* som kan settes aktiv av de strategiene som er benyttet. Denne strategien blir aktiv når det er kommet en observasjon av en flaske eller en boks inn til planleggingssystemet. Strategien er implementert som en tilstandsmaskin, og selve tilstandsmaskinen fungerer som tiltenkt.

Før avreise til Frankrike var det ikke nok tid til å teste ut mer enn den grunnleggende tilstandsmaskinen, og uforutsette hendelser ble ikke tatt høyde for. Det er blant annet ikke implementert håndterer som detekterer om flasken eller boksen ikke blir plukket opp. Det er derimot implementert håndterer som setter nytt waypoint til sorteringsboksene, om det er plukket opp en flaske, når

planleggingsystemet i utgangspunktet tror det har hentet en boks. Deteksjonen av opplukket flaske eller boks gjøres av bildebehandlingsystem to, som er tilknyttet kamera inne i roboten. Det sendes melding fra planleggingsystemet til bildebehandlingssystem to, når nytt objekt er plukket opp. Bildebehandlingssystemet sender da en melding tilbake som forteller om der er flaske, boks eller ingenting. Når bildebehandlingsystem to detekterer ingenting, er det som sagt ikke implementert håndterer. Roboten snur seg uansett mot boksen og skyter, før den snur seg tilbake mot midten av bordet og gir melding om at strategien er ferdig. Når strategien er ferdig setter *PlansysObservation* strategien igjen til inaktiv. Håndtereren som tar seg av setting av nytt waypoint når annet objekt enn det som i utgangspunktet var forsøkt plukket opp, er testet. Denne funksjonaliteten er testet ved å legge en flaske foran roboten. Roboten starter å kjøre mot denne samtidig som flasken blir byttet ut med en boks. Roboten plukker opp boksen og kjører til boksbeholderen og sorterer riktig. Denne feilkorreksjonen er testet ved flere anledninger både for boks og flaske.

Feilkorreksjon skulle også vært lagt inn der bildebehandlingsystemet gir melding om en flaske eller boks som ligger utenfor bordet. Dette er det ikke tatt høyde for da planleggingsystemet uansett snur roboten inn mot midten av bordet etter å ha skutt, og før den tar bilde. Denne feilkorreksjonen kan legges inn da strategien til en hver tid vet hvor roboten befinner seg, og vinkelen denne har. Størrelsen på bordet er også kjent, slik at det er lett mulig å regne ut om avstanden til et nytt observert objekt er utenfor bordet. Er objektet utenfor bordet, bør strategien sette et avbrudds-waypoint med vinkel inn mot midten av bordet i strukten *interrupt_control_struct*, samtidig som strategien returnerer negativ verdi på *getAgentPriority*.

BatterySort
Strategien er implementert, men benyttes ikke på roboten, da sorteringsenheten for batterier ikke ble ferdigstilt.

MotorAi
Før avreise til Frankrike ble denne strategien implementert til å stoppe roboten for hindringer. Dette var tilfredsstillende for kvalifiseringen av roboten. Under konkurransen ble det implementert avbruddskjøring av waypoint på roboten. Avbruddskjøringen gjør at roboten kommer seg unna roboter som den er på vei til å krasje i. Selve unnamanøveren fungerer fint, men her kan det legges inn mer funksjonalitet ved å kombinere sensorene. Det kan også være greit å ta høyde for tiden mellom hver kollisjonsdeteksjon, og basert på denne avgjøre robotens handling sammen med sensorene. Det er viktig i denne strategien å ta høyde for at et avbrudds-waypoint ikke er utenfor bordet. Strategien vet til en hver tid hvor roboten befinner seg på bordet og kan ut fra dette bestemme hvor den skal sette waypoint når kollisjonsmelding mottas.

Alle feil som er observert når det gjelder planleggingsystemet kan knyttes til strategiene. Roboten var ikke ferdigstilt før dagen før avreise til Frankrike, så tiden for å teste ut strategiene holdt ikke for å få strategiene robuste.

Funksjoen *GetAgentPriority* i objektet *PlansysObservation* har et stort potensiale. Slik funksjonen nå er implementert vil den gjøre strategien med høyest prioritet aktiv, og beholde denne aktiv til den er ferdig utført. Ved å skrive om algoritmene i denne strategien, kan man blant annet kombinere strategier. Man kan også implementere funksjonen, slik at en strategi ikke nødvendigvis må gjøre seg ferdig før det kan velges en ny strategi.

6.3.5 Navigasjonssystemet

Navigasjonssystemet ble utviklet av Kristian M. Knausgård. Det ble brukt mye tid av undertegnede sammen med Kristian for å samkjøre motordrivere, Proxy-/Gateway-systemet og navigasjonssystemet, tidlig i vårsemesteret. Systemet fungerte fint etter at samkjøringen ble ferdigstilt, helt til det oppsto feil på ene motordrivermodulen, som beskrevet i kapittel 4.2.2. Da denne feilen oppsto, ble parametrene i regulatoren justert for å se om feilen kunne tilskrives denne. Dette ble avkreftet da feilen senere ble lokalisert til en H-bro.

Da konkurransen nærmet seg og roboten begynte å bli ferdigstilt, var det igjen behov for navigasjonssystemet. Regulatorparametrene var da fremdeles ikke riktig innstilt. Undertegnede sendte da waypoint fra planleggingssystemet til navigasjonssystemet, samtidig som forskjellige regulatorparametre ble justert. Akseptable parametre ble etterhvert funnet, og navigasjonssystemet fungerte deretter fint.

6.3.6 Bildebehandlingssystemet

Bildebehandlingsystemet på roboten består av to kamera. Det viste seg tidlig at med så mange flasker og bokser som til en hver tid finnes på bordet, er sannsynligheten for å ta opp en flaske istedenfor en boks stor, selv om roboten kjører etter riktig type i utgangspunktet. Det ble montert et kamera inne i roboten som planleggingssystemet skulle benytte til å korrigere for dette. Begge kamerasystemene har fungert utmerket, både med tanke på riktig informasjon og hastigheten på en forespørsel. Grensesnittet mellom de to kamera og planleggingssystemet har basert seg på den samme strukten, da all informasjon bildebehandlingsystemet skal gi er posisjoner og type observasjon.

6.3.7 Proxy- /Gateway-systemet

Proxy-/Gateway-systemet benyttes til å sende meldinger til alle modulene i systemet. En melding kan sendes til flere moduler på en gang, slik som start og stoppmeldinger som sendes både til navigasjonssystemet og til CAN-bussen.

Proxy-/Gateway-systemet inneholder grensesnittet mot CAN-bussen og transformerer meldinger fra Posix-meldinger til CAN-meldinger og motsatt.

Ved hjelp av dette programmet trenger de andre programmene ikke ta høyde for hvilken kommunikasjon som benyttes mot modulene utenfor hoveddatamaskinen. Skulle man ønske å bytte ut CAN-bussen med en annen type buss, vil ikke resten av programmene på hoveddatamaskinen påvirkes.

Det har blitt opprettet egne driverfiler for moduler tilknyttet CAN-bussen som styres fra programmer på hoveddatamaskinen. Dette er filene *ProxyMotorControll.cpp*, og *ProxyCollisionDetection.cpp*. Ved å opprette egne driverfiler for hver modul, vil systemet bli mer oversiktig og modulen kan enkelt byttes ut sammen med denne filen. For roboten er det opprettet egne filer for motordrivermodulene og antikollisjonsystemet. De resterende systemene baserer seg på kommando meldinger, og har ikke egne filer. Kommandomeldinger er samlet i filen *proxyCommands.cpp* når meldingen kommer fra posix-køene, og i filen *can_buss.cpp* når kommandoen kommer fra CAN-bussen.

Ved å studere dette systemet sammen med filen $proxyconfig.h$, og $posix\_message\_config.h$, vil man få en god oversikt over hvilke meldinger som sendes i systemet og hvor de blir sendt. Meldinger mellom bildebehandlingsystemet og planleggingsystemet blir ikke sendt gjennom dette proxysystemet, men sendes direkte. Mellom navigasjonssystemet og planleggingsystemet blir start og stoppmeldinger sendt gjennom dette systemet da disse skal sendes til flere moduler på en gang. De resterende meldingene mellom disse to programmene sendes direkte.

Proxy-/Gateway-systemet har fungert utmerket ved benyttelse, men sett i ettertid kunne det derimot vært implementert mer dynamisk. Ved å opprette vektor-lister i Proxy-/Gateway-systemet som inneholder hvilke moduler som skal motta gjeldene melding, hadde dette kunne blitt oppnådd. Proxy-/Gateway-systemet sender da en melding til alle moduler som ligger i denne listen for hver melding det mottar. Man kan da opprette en egen melding som inneholder en strukt med to variable, som hver enkelt modul skal kunne sende til Proxy-/Gateway-systemet. Første variabel i strukten skal inneholde identifikatoren til modulen som sender meldingen, neste variable inneholder meldingsidentifikatoren gjeldene program eller modul ønsker å motta. Når Proxy-/Gateway-systemet mottar en slik melding, skal modulen som sendte meldingen legges til i vektorlisten på tilhørende meldingsidentifikator.

Undertegnede anbefaler fortsatt bruk av et Proxy-/Gateway-system som dette, men da gjerne med oppgradering til et mer dynamisk system, som anbefalt over.

6.4 Mekanisk konstruksjon

6.4.1 Chassis

Et av de største problemene Eurobotlagene fra NTNU har stått ovenfor er graden av gjenbrukbare deler som har vært tilgjengelig. Dette har det siste året værttatt høyde for i alle ledd også inbefattet den den mekaniske konstruksjonen av chassis til roboten. Da reglene for størrelsen på roboten har vært de samme hvert år har årets robot blitt bygget for å komme opp mot denne størrelsen. Roboten har fått en omkrets på 119 centimeter, hvor 120 centimeter er maksimal tillatt størrelse.

Roboten er bygget over to plan. Ved å plassere mesteparten av elektronikken på øvre plan, har hele nedre plan vært dedikert mekaniske enheter for å løse robotens oppgave. Dette året har plassen vært viet til flaske-/boks-sortereren, men ved å fjerne denne kan en annen enhet lett settes inn.

Roboten er bygget for å lett kunne fjerne front/topplaten, og sidedekslene på roboten med noen få skruer. Med disse dekslene fjernet, har man full tilgjengelighet til alle robotens moduler.

Den mekaniske konstruksjonen av chassis er både solid og har et bra utseende.

Da neste års oppgave ble presentert under konkurransen i Frankrike, ble det bekreftet at årets robotstørrelse blir som i år. Oppgaven går ut på å sortere innebandyballer av forskjellig farge. Det vil si at store deler av årets robot kan gjenbrukes til neste år. Roboten har på nedre plan plass til flere innebandyballer inne i roboten. Det eneste som må gjøres er å bytte ut sorteringsenheten og mest sannsynlig bunnplaten. Undertegnede anbefaler på det sterkeste å benytte så mye som mulig av den mekaniske konstruksjonen på neste års robot, da dette vil frigjøre mye tid som kan benyttes på andre oppgaver. Det kan være fristende å designe et nytt chassis for et nytt Eurobotlag, da det i utgangspunktet ser ut som en lett jobb. Det anbefales igjen på det sterkeste å gjenbruke gjeldende chassis da konsruksjon av et nytt tar betydelig mer tid enn det kan se ut til. Chassis sammen med motorer og gir trengs også for utprøving av programmer.

6.4.2 Motor-/Gir-moduler

Drivhjul og motorblokkene har fungert tilfredsstillende sammen med løpehjulene. Det ble derimot observert at drivhjulene av og til løftet seg litt fra bakken og ble stående å spinne. To lodd ble festet fremme på roboten for å kompensere for dette. Hovedgrunnen til spinningen ble lokalisert til støttehjulene bak. Ved at gummihjulene som berører bakken på støttehjulene ikke er festet rett under sentreringstappen, vil disse gi forskjellig trykk mot underlaget alt ettersom hvilken posisjon de står i. Braketten som holder støttehjulet inneholder også et kulelager, hvor sentreringstappen til støttehjulet er festet. Dette kulelageret var i utgangspunktet ikke festet godt nok, slik at disse flyttet seg oppover i

braketten etterhvert som roboten ble bygget og fikk mer tyngde. Dette førte til at ene støttehjulet ble stående lenger oppe enn det andre, og var medvirkende til at ene drivhjulet løftet seg fra underlaget.

Løsningen ble å feste kulelageret med en ny brakett slik at det ikke skulle skli oppover. Sammen med loddene fremme på roboten fungerte dette tilstrekkelig.

På grunn av plasseringen av hjulet som berører bakken på støttehjulene ikke er sentrert under sentreringstappen, blir det et moment om denne tappen. Dette momentet fører til at hele hjulbraketten bøyer seg, som igjen kan føre til at de to støttehjulene kan få forskjellig høyde.

O-ringen på løpehjulene er det viktig å vaske med vann og holde ren, slik at den ikke sklir mot underlaget. O-ringen er også viktig å kontrollere og bytte ut da den fort blir tørr og sprekker. Dette fører igjen til at løpehjulene kan skli mot underlaget.

Gjennbruk av drivhjulmodulene og løpehjulene anbefales. Når det kommer til støttehjulene bak, kan disse også gjennbrukes, men da anbefales det å stive disse opp med en plate mellom hjulet og kulelageret, slik at hjulbraketten ikke bøyer seg. Undertegnede anbefaler derimot å bytte disse til støttehjul som har kulehjul under sentreringstappen, for å være sikker på at det er samme trykket på alle hjul til en hver tid. Slike hjul finnes det mange varianter av, og kan blant annet bestilles fra «Mapp Caster & Supply» (Caster, 2007). Bilde av slike hjul vises i figur 6.1.

Figur 6.1: Støttehjul

6.4.3 Antikollisjonssensorer

Med tre antikollisjonssensorer plassert i front av roboten, to på sidene 20 centimeter over bakken og en i midten 25 centimeter over bakken, har roboten vært i stand til å detektere hvor en kollisjon kan oppstå. Man kan plassere disse sensorene alt etter hvor man vi detektere kollisjon. De kan rettes nedover, oppover eller til sidene. Retter man sidesensorene mot midten av kjøreretningen, vil man kunne benytte en avveining mellom disse hvis der er oppstått en kolli-

sjon. Plasseringen av sensorene rett fremover i posisjonen de nå har på roboten, detekterer fint når en annen robot kommer i kollisjonsposisjon.

6.4.4 Kamera

Kameraene på roboten ble plassert i gode posisjoner for å utføre tiltenkt oppgave. Kamera i front er plassert rett over sorteringsenheten slik at programmene ikke trenger å ta høyde for akseforskyvninger av observasjoner av flasker og bokser. Kamera inni roboten ble plassert på en ledig plass der det hadde oversikt over hele armen til flaske-/boks-sortereren. Det er ikke observert problemer med plasseringen av kameraene.

6.4.5 Koblingsmodul

Koblingsmodulen på roboten har både betjeningspanel for brytere, sikringer og spenningsforsyning med både fem og 12 volt. Modulen fordeler spenningen fra batteriene, gjennom brytere sikringer og kontakter ut til de forskjellige modulene på roboten. Motordriverne har egne kontakter for tilførselsspenning i front av denne. Det har vært en klar fordel å samle alle kontakter på en plass. Skal man koble fra en modul, trekker man ut kontaktene på denne og fjerner den. Dette hadde vært verre om ledningene hadde blitt trukket rundt hele roboten uten struktur. Når man skal starte roboten før et spill i konkurransen, er det også greit at alle brytere er samlet på en plass, slik at man straks ser om en bryter ikke er på. Det kan være fort å glemme en bryter man ikke ser når man skal starte alle robotens programmer. En skal også passe på at alt er riktig innstilt på de tre minuttene man har til klargjøring av roboten før kamp.

Modulen er utvidbar, da den har plass til flere kontakter i front, og flere brytere, lysdioder og sikringer oppå. Der finnes en USB kontakt på modulen. Det anbefales å montere en nettverkskontakt under denne. På årets robot har nettverkskontakten direkte på hovedkortet vært benyttet, noe som har vært litt kronglete, da dette er montert på innsiden av koblingsmodulen. Passende nettverksledning er Eurobotlaget i besittelse av fra 2006 roboten.

Kapittel 7

Resultater og Diskusjon Prosjektledelse

Frem til og med desember 2006, hadde Kristian M. Knausgård og undertegnede hovedansvaret sammen for utvikling av roboten. Fra Januar 2007 fikk undertegnede alene hovedansvaret for utviklingen og ferdigstillelsen av roboten, som prosjektleder. Lars Vråle og Eksperter i Team gruppen kom inn i prosjektet fra og med vårsemesteret 2007. Lars har hatt ansvar for utvikling av robotens bildebehandling, og Eksperter i Team gruppen har utviklet sorterings og skytemekanismen for flasker og bokser.

I Januar og begynnelsen av februar brukte Kristian og undertegnede en del tid på å ferdigstille en kjørbar grunnbase for roboten. Navigasjonssystemet og første motordrivere ble ferdigstilt. Fra denne tid har ikke Kristian hatt anledning til samme deltagelse som tidligere.

Lars har ikke hatt kontor i tilknytning til undertegnede. Det har ikke vært noen fordel for prosjektet og samarbeidet. Det har ført til at Lars og undertegnede har arbeidet mye alene, og ikke som en gruppe, noe som ville vært naturlig om man hadde hatt samme kontor. Etter at det grunnleggende på roboten begynte å komme istand og bildebehandlingsystemet skulle testes, har Lars tilbragt mye tid ved spillebordet for å få riktig bakgrunn for bildebehandlingen. Lars fikk da en mer naturlig deltagelse i prosjektet som helhet enn tidligere.

Etter Lars kom mer aktivt inn i prosjektet, foruten bildebehandlingen, kunne han lettere delta i forefallende arbeid. Oppgaver påbegynt av Kristian og undertegnede på et tidligere tidspunkt, ville etter undertegnedes mening vært lite inspirerende for Lars å ferdigstille. Om Lars skulle delta i forefallende arbeid på tidspunkt før dette måtte dette planlegges fordi undertegnede holdt til i D og B blokka, og Lars i G blokka. D og B blokka ligger i motsatt ende av bygget i forhold til G. Mange av oppgavene som skulle gjøres var små oppgaver som gjerne ikke tok mer enn en halvtime. Skulle Lars deltatt på disse ville det gått mer tid, fordi det ville krevd mer planlegging.

Eksperter i Team gruppen utviklet i løpet av vårsemesteret 2007, sorteringsmodulen for flasker og bokser. Arbeidet med denne har stort sett vært utført onsdager, og undertegnede har etter beste evne prøvd å være tilgjengelig for gruppen disse dagene. Å designe et godt og robust mekanisk system for en slik sorteringsmodul, er krevende. Utviklingen av systemet tar mye tid og trenger mye testing. Dette medførte at systemet ikke ble ferdig før samme uke som avreise til konkurransen. For Eksperter i Team gruppen nærmet eksamen seg. Prosjektarbeidet kunne derfor ikke vies full oppmerksomhet, det ville gått på bekostning av de andre fagene.
Årets Eksperter i Team gruppe, designet et bra sorteringsystem. Systemet hadde likevel en rekke feil som måtte ordnes før systemet skulle virke tilfredsstillende. Undertegnede brukte mye tid som var avsatt til andre formål for å rette opp disse feilene. Sorteringen var hovedoppgaven til roboten, og fungerte ikke dette systemet hundre prosent ville ikke deltagelse i Eurobot vært mulig. Undertegnede vil anbefale for senere år å designe mekaniske enheter som er grunnleggende for å løse oppgaven så tidlig som mulig. Designet og konstruksjonen bør da gjøres på høstsemesteret, slik at dette er klart for montering i roboten når vårsemesteret begynner. Da kan hele semesteret benyttes til forbedring av løsningen, og andre systemer som er avhengig av eller bruker dette, kan teste funksjonaliteten på et tidlig tidspunkt.

Som prosjektleder og eneste person som jobbet med prosjektet på fulltid, førte det til at undertegnede måtte tilpasse seg de andre deltagernes tidsplan. Dette medførte et stadig bytte av fokus, og avbrytelser fra jobbing med egne delsystemer.

Undertegnede vil anbefale for senere års konkurransedeltakere, at Eurobotlaget settes sammen på høstsemesteret. Denne sammensetningen bør være fast helt frem til konkurransen. Dette vil føre til et kollektivt ansvar for hele prosjektets gjennomføring, og en eierskapsfølelse for roboten. Ved å komme senere inn i prosjektet kan det være vanskeligere å få gjennomslag for egne ideer, da retningslinjene allerede er lagt. Eurobot laget bør etter undertegnedes mening bestå av minimum fire personer. Disse bør ha både prosjektet og masteroppgaven på roboten.

Kapittel 8

Diverse

8.1 Kretskortdesign

Kretskortene som er designet til roboten er designet i CAD programmet Eagle fra CadSoft (CadSoft, 2007). Det er designet en god del komponenter for dette programmet, som ikke finnes standard i Eagle eller på Internett. Komponentbibliotekene som er laget finnes på vedlagt CD.

I kjelleren på instituttet finnes et rom som benyttes til produksjon av printkort. Nøkkelen til dette rommet har Eurobotlaget disponert. Dette fører til at alle studenter som skal etse printkort må hente denne nøkkelen på Eurobotkontoret. Eurobotdeltagerne må av denne grunn stadig gi en opplæring i produksjon av printkort. Undertegnede har laget en kort bruksanvisning for etsing av printkort, som finnes i vedlegg F.

8.2 Anbefalinger til NTNU for Eurobot videre

Undertegnede har deltatt i Eurobot to år på rad, første år som medhjelper for å se og lære, andre år som prosjektleder og deltaker. Under denne perioden har undertegnede gjort seg opp en del bemerkninger og synspunkter på hvordan NTNU skal kunne hevde seg i Eurobot i fremtiden. Hvert år har det vært bygget stort sett helt nye roboter, og mye ny elektronikk. Lagene har vanligvis bestått av to personer som tar masteroppgave og prosjektoppgave på roboten. Noen år har en fjerdeklasse gruppe fra Eksperter i Team hatt en deloppgave til roboten. Mye av det mekaniske er satt ut til Kybernetikk verkstedet. Dette har ført til at stort sett hele roboten og de fleste delsystemene er konstruert av de to hovedoppgave studentene. Det administrative ved en slik oppgave tar også en stor del av tiden. Når man studerer andre lag som deltar i Eurobot, og da særlig de som hevder seg, ser man at det kan være fra 10 til 20 deltakere på hvert lag, der de har ansvaret for hver sine oppgaver.

Undertegnede vil anbefale Institutt for Teknisk Kybernetikk om å gjøre Eurobot mer tverrfaglig på NTNU der man går inn for følgende sammensetning.

- En student fra «produktutvikling og produksjon» eller «Institutt for produktdesign» til den mekaniske konstruksjonen.
- En eller to student som tar seg av strategiplanleggingssystemet, og bildebehandlingen. Disse studentene kan komme fra «Institutt for Teknisk Kybernetikk» eller fra «Institutt for datateknikk og informasjonsvitenskap».
- En student fra «Institutt for Teknisk Kybernetikk» som tar seg av navigasjonssystemet, og pådragssystemet. Der denne har ansvaret for posisjoneringen og forflytningen av roboten.
- En student fra «Institutt for Teknisk Kybernetikk» som tar seg av å løse årets oppgave sammen med studenten som er ansvarlig for den mekaniske konstruksjonen. Denne studenten må tilpasse elektronikk og programmer for oppgaven. Studenten bør også jobbe med andre mindre moduler som roboten kunne trenge.
- En student som tar seg av prosjektstyringen, der fokus det er rettet stor fokus på at grensesnittet mellom de forskjellige deler blir overholdt, og framdriftsplanen overholdes. Denne studenten tar seg også av all kontakt med Eurobot komiteen og sponsorer. Studenten kan også designe mindre moduler for roboten, og annet forefallende arbeid.

For å lykkes med en slik sammensetning, er det veldig viktig med samarbeid mellom de forskjellige deltagerne, selv om disse har klart definerte oppgaver som de skal utføre. Ved at en student får konsentrere seg om mindre delsystemer av roboten vil flere av disse systemene kunne designes robuste, godt dokumenterte og bli helt ferdig før testperioden for roboten begynner.

Det anbefales på det sterkeste å fortsette bruken av CAN-buss og posix-køer på robotene, da dette gjør roboten modulbasert, og hver enhet får et klart definert grensesnitt. Modulene bør designes gjenbrukbare for senere års roboter, uten at dette medfører større forandringer. Man vil da på sikt når man har gode systemer for bildebehandling, planleggingssystem og navigasjonsystem ferdig, samt flere hardware moduler tilgjengelig. Da vil man kunne ha færre studenter på laget for å hevde seg i konkurransen.

Mest mulig av den mekaniske konstruksjonen, da innbefattet mekanisk enhet for å løse oppgaven, bør ferdigstilles før jul. Da vil man ha hele vårsemesteret til å utvikle programvare, og eventuelle feil er lett å oppdage.

Undertegnede var i kontakt med Keith Downing på «Institutt for datateknikk og informasjonsvitenskap», før implementasjonen av planleggingsystemet på årets robot, se vedlegg E. Downing virket interessert i muligheten for et

samarbeid på tvers av instituttene. Det anbefales å å kontakte ham for å rekruttere IDI studenter til Eurobot.

8.3 Spillebordet

Bygging av spillebordet ble benyttet til en bli kjent kveld for undertegnede, de to andre masterstudentene og Eksperter i Team-gruppen. Undertegnede gjorde alt klart for maling av bordet og bygging av sorteringsbokser. Spillebordet fra fjordåret hadde hull for bordtennisballer, se regler Eurobot 2006 (Eurobot Commitee, 2005). Disse hullene ble tettet ved hjelp av propper i tre og sparklet. Sorteringsboksene ble laget av trebjelker og kryssfiner.

Lokalet rundt spillebordet blir stadig benyttet til omvisninger, så anledningen ble også benyttet til å rydde lokalet. I løpet av kvelden ble det servert pizza og drikke på Eurobots regning, for å få en best mulig ramme rundt situasjonen.

8.4 Omvisninger

Ved instituttet er det stadig omvisninger for blant annet videregående skoler. Det er også andre som kan være interessert i å se hva som foregår på instituttet. Det er vanlig at Eurobot stiller opp for å vise frem roboten, vise filmer fra Eurobotkonkurransen og prate om studiet og byggingen av roboten. Det har høstsemesteret vært holdt en rekke presentasjoner. Disse har blant annet vært representert:

`Katedralskolen`
22.februar var en gruppe elever fra katedralskolen på omvisning på NTNU. Flere av de fremmøtte kom fra en gruppe som kaller seg «Robotgjengen på Katta», og hadde deltatt i RoboCup Junior, blant annet i Tyskland i 2006. Det var da naturlig at de fikk en informasjon om Eurobot og NTNU sin deltagelse i denne konkurransen. Det ble vist video og flere Eurobotroboter. Robotgjengen har hjemmeside på internett (Robotgjengen, 2007).

`Gausdal Videregående skole`
Gausdal videregående skole var på omvisning på NTNU 8. mars. Her var det en gruppe på 20 personer som var delt opp i to grupper. Begge gruppene fikk en 20 minutters presentasjon av Eurobot.

`Abelfinale Deltagere`
9. mars var finaledeltagerne i Abelkonkurransen på omvisning. Abelkonkurransen er en konkurranse i matematisk problemløsing for videregående skole. Abelfinalen har egen hjemmeside (Abelkonkurransen, 2007).

`Jentedag på Elektronikk og Kybernetikk`
For å få kvinneandelen på elektronikk og kybernetikk opp, ble det arrangert jentedag på disse fakultetene 12. mars. Her var det jenter fra videregående skoler fra hele landet som hadde møtt opp. De fremmøtte var delt inn i åtte grupper som skulle ha omvisning på tilsammen åtte poster. En av postene var da Eurobot. Undertegnede og Lars Vråle stilte for Eurobot og holdt korte presentasjoner for de fremmøtte. Hele presentasjonen varte over to ganger en time.

`Lillehammer Videregående skole`
Undertegnede holdt 15. mars en presentasjon av Eurobot for tre grupper fra Lillehammer videregående skole. Denne presentasjonen tok til sammen en time. Her var det flere deltagere som syntes kybernetikk virket som et interessant studie.

8.5 Litteraturstudie og læringsandel

Læringsandelen ved å implementere planleggingsystemet, har vært stor. Undertegnede hadde før prosjektet ikke programmert C++, og har i løpet av prosjektet lært en rekke lure måter å programmere objektorientert. Ved hjelp av boken «Design Patterns» (Gamma, E. ,Helm, R. , Johnson, R. ,Vlissides, J., 1995) har det vært studert mange designmetoder for å løse konkrete programmeringsproblemer.

Flere bøker og rapporter «papers» innen kunstig intelligens og planleggingssystemer er studert. Boken «Introduction to AI Robotics» (Murphy, 2000) har vært særdeles interessant da denne beskriver flere forskjellige typer planleggingsystemer for roboter. Boken gir også en fin introduksjon til robotnavigering.

Det å være prosjektleder har gitt innsikt i hvor mye tid koordinering, og oppdatering av deltagere i et prosjekt tar. Om et grensesnitt ikke er blir overholdt i en modul kan dette medføre mye ekstraarbeid. Av denne grunn er det viktig at prosjektleder er sikker på at alle grensesnitt overholdes. Man må også ofte avbryte eget arbeid for å oppdatere andre deltagere, som må forsikre seg om at deres system oppfyller krav og grensesnitt.

Prosjektleder må også passe på at tidsfrister overholdes, slik at der er tid til eventuell feilretting. En oppgave tar også uansett hva type oppgave det er lenger tid en man forventer i utgangspunktet.

Ved å bygge roboten modulbasert ved hjelp av CAN-buss, har undertegnede ikke bare sett fordelen av et meldingsbasert system, men også fått god kjennskap til CAN-bussen, godt utover det som er implementert på roboten.

Kapittel 9

Konklusjon

Som oppgave tittelen tilsier, har det i denne masteroppgaven blitt konstruert en autonom robot, som har deltatt i Eurobot Open 2007. Flere personer har vært involvert i utviklingen av moduler til roboten, og undertegnede har som prosjektleder hatt hovedansvaret for ferdigstillelse av roboten.

Fra første stund i utviklingen av roboten har det vært lagt vekt på å bygge roboten modulbassert. Dette er oppnådd ved bruk av CAN-buss, og internt på robotens hoveddatamaskin har programmene kommunisert ved hjelp av posix-køer. Ved å benytte disse to systemene har hver modul fått et klart definert grensesnitt basert på meldingssending. CAN-buss-modulene har i tilegg fått levert spenning til logikkkretsene, gjennom CAN-bussledningen.

Når det gjelder planleggingsystemer på tidligere NTNUs Eurobot-roboter, har lite vært gjort. I denne oppgaven er det utviklet et rammeverk, som kan inneholde et stort antall forskjellige strategier for roboten. Rammeverket baserer seg på «Action-Selection» prinsippet, som benytter en algoritme til å velge ut en av mange strategier basert på strategienes momentane prioritetsverdi. Alle meldinger som kommer inn til systemet, som bildeobservasjoner og posisjons oppdateringer, vil automatisk oppdatere alle strategier i systemet. På denne måten har alle strategier siste informasjon om verdenen de skal operere i. Rammeverket begrenser seg ikke bare til bruk på roboter, men kan også implementeres i andre systemer som trenger strategiplanlegging.

Motordrivermodulen versjon 2.2 som ble utviklet under prosjektperioden høsten 2006, ble videreutviklet til en versjon 3.0 i denne masteroppgaven. Det ble laget to nye motordrivermoduler. Den siste versjonen ble oppgradert med diverse filter, som fjernet støy generert fra blant annet motorene og H-broene.

Gjennom å bygge roboten modulbasert håper undertegnede at NTNU på sikt skal kunne vinne Eurobot Open. De beste lagene i Eurobot bruker hvert år oppigjen mest mulig deler fra tidligere år. NTNU har nå flere gode moduler til bruk som kan benyttes senere år.

Chassis på roboten er i år bygget maksimal størrelse som er lov i henhold til reglementet. Denne størrelsen er hvert år lik, så neste års Eurobotlag kan

spare mye utviklingstid på gjenbruk av årets chassis. Chassis er også bygget for rask å kunne demonteres for å komme til de interne moduler.

Resultatet fra konkurransen ble ikke like bra som man i utgangspunktet hadde håpet. NTNU laget endte på 25 plass av 39 lag. Dette skyldtes i hovedsak uforutsette hendelser i ukene før avreise til Frankrike. Merarbeidet disse uforutsette hendelsene ga resulterte i mindre tid til testing av hele systemet samlet, enn nødvendig for å få et robust system.

Bibliografi

Abelkonkurransen (2007). Abelkonkurranansen. http://abelkonkurransen.no.

Atmel (2006). *AT90CAN128 Manual*.

Atmel (2007). AVR Studio 4. http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725.

Brown W., Malveau R., McCormick H., Mowbray T. (1998). *Anti Patterns*. John Wiley & Sons Inc.

Bulgin (2007). Bulgin. http://www.bulgin.co.uk.

CadSoft (2007). Cadsoft. http://www.cadsoft.de.

Caster, M. (2007). Mapp caster. http://www.mappcaster.com/ProdInd/Ball-Transfer-Index.aspx.

ClasOhlson (2007). Neodym-magneter. http://www.clasohlson.no/Product/Product.aspx?id=390590.

Commell (2007). Commell technical support centre. http://www.commell.com.tw/Support/SBC/LV-677(S).htm.

Eurobot Commitee (2005). Regler, Eurobot 2006. http://www.eurobot.org/eng/archives_2006/docs/E2006_rules.pdf.

Eurobot Commitee (2006). Regler, Eurobot 2007. http://www.eurobot.org/eng/docs/2007/E2007_rules_%20definitive%20version.pdf.

Eurobot Commitee (2007). FAQ3, Eurobot 2007. http://www.eurobot.org/eng/docs/2007/E2007_rules_FAQ3.pdf.

Fossen, Thor I. (2002). *Marine Control Systems*. Marine Cybernetics AS.

Gamma, E. ,Helm, R. , Johnson, R. ,Vlissides, J. (1995). *Design Patterns*. Addison-Wesley.

Garsjo, Sondre og Platou, Halvor (2006). Eurobot 2006. Master's thesis, Norges teknisk-naturvitenskapelige universitet.

Heyes-Jones, J. (2007a). A star heyes-jones code project. http://code.google.com/p/a-star-algorithm-implementation.

Heyes-Jones, J. (2007b). A star justin heyes-jones. http://www.geocities.com/jheyesjones.

Intel (2007a). Intel 82573 Gigabit. http://www.intel.com/design/network/products/lan/controllers/82573.htm.

Intel (2007b). Intel Proadmin. http://downloadcenter.intel.com/filter_results.aspx?strTypes=all&ProductID=2196&OSFullName=Windows*+XP+Home+Edition&lang=eng&strOSs=45&submit=Go%21.

Keith Downing (2007). Keit Downing. http://www.idi.ntnu.no/~keithd/iweb/Site/Homepage.html.

Kjemphol, Gunnar og Knausgaard, Kristian M (2006). *Prosjektoppgave Eurobot 2007*. Norges teknisk-naturvitenskapelige universitet.

Liao C., Yo R., Chen Z., Chan D., Fu L. (2005). *Behavior Injector: An Architectural Pattern for Rapid Prototyping the Reactive Intelligent Robots*. National Taiwan University.

Mekanex (2006). Mekanex Maskin AB. http://www.mekanex.se/.

Midt-Norsk Handelskammer (2007). Midt-norsk handelskammer. http://carnet.chamber.no/trondheim.

Murphy, R. R. (2000). *Introduction to AI ROBOTICS*. Massachusetts Institute og Technology.

National Semiconductor (2005). *LMD18200 3A, 55V H-Bridge*.

Pattie Maes (1989). *How to do the right thing, MIT AI-Memo 1180*. Massachusetts Institute of Technology.

PEAK (2007a). Peak. http://www.peak-system.com/db/gb/pcanusb_gb.html.

PEAK (2007b). Peak linux driver. http://www.peak-system.com/linux.

Robotgjengen (2007). Robotgjengen katta. http://robocup.idi.ntnu.no/wiki/index.php/Robogjeng_p%C3%A5_Katta.

Rodney A. Brooks (1985). *A Robust Layered Control System for A Mobile Robot*. National Taiwan University.

Semiconductor, N. (2002). National semiconductor. http://www.national.com/appinfo/power/files/f5.pdf.

SRI (1970). Sri strips. http://www.ai.sri.com/pubs/files/tn043r-fikes71.pdf.

SRI (2007a). Sri interational. http://www.ai.sri.com.

SRI (2007b). Sri shakey. http://www.sri.com/about/timeline/shakey.html.

Stan Franklin (1995). *Artificial Minds*. MIT Press.

STMicroelectronics (2007). L7805. http://www.st.com/stonline/products/literature/ds/2143.pdf.

TracoPower (2006a). Datablad traco power ten15wi. http://dsb.tracopower.com/upload/dsbuserfile/cpn_tracopower/0_ten15wi.pdf?

TracoPower (2006b). Datablad traco power ten30wi. http://dsb.tracopower.com/upload/dsbuserfile/cpn_tracopower/0_ten30wi.pdf?

TracoPower (2007). Traco power. http://www.tracopower.com.

UnitedHobbies (2007). United hobbies. https://www.unitedhobbies.com/UNITEDHOBBIES/store/comersus_viewItem.asp?idProduct=2.

Wikipedia (2007). Wikipedia a-star. http://en.wikipedia.org/wiki/A%2A.

Tillegg A

BIOS-oppgradering

Oppgradering av BIOS på Commell LV-677 gjøres på følgende måte:

1. Lag en DOS disket det kan som kan boote. Dette gjøres på en Windows PC under «My Computer» ikonet. Velg «Format» og formater system-diskett.

2. Last ned siste BIOS-versjon, og programmet for flashing «awdflash.exe» fra Commells hjemmeside (Commell, 2007).

3. Legg BIOS og «awdflash.exe» på en diskett.

4. Koble til hovedkortet bare tastatur, skjerm og diskettstasjon. Koble fra eventuelle harddisker og Compact Flash-kort.

5. Boot hovedkortet ved hjelp av boot disketten, og sett deretter inn disketten med «awdflash.exe». Start programmet «awdflash.exe».

6. Du får nå spørsmål om filnavnet fil den nye BIOS. Skriv inn dette og trykk Enter. File Name to Program: 67714.bin

   Du vil nå få spørsmål om du ønsker å lagre den gamle BIOS versjonen. Skriv eventuelt inn navnet du ønsker på denne, og trykk Enter. Du får da spørsmål om du ønsker å programmere, trykk da «Y». Nå vil BIOS bli programmert, og det er veldig viktig å ikke skru av maskina når dette pågår. VIKTIG: Det er veldig viktig å ikke skru av maskina når BIOS-oppgradering pågår.

7. Når BIOS er ferdig oppgradert fjernes disketten og man trykker F1. Maskinen vil nå restarte og man kan koble til eventuelle diskettstasjoner man har fjernet.

8. Det er nå viktig å legge inn ønskede BIOS innstillinger. For LV-677 med Compact Flash-kort får man ikke lastet opperativsystemet uten å sette «On-Chip Serial ATA» til «Disabled». Denne innstillingen finner man under: Integrated Peripherals -> OnChip IDE Device

Tillegg B

Installere Ubuntu på COMMELL LV-677 FlashDisk

Installasjon av ubuntu gjøres ved hjelp av minnepinne som beskrevet i Eurobot Prosjektrapport 2006 (Kjemphol, Gunnar og Knausgaard, Kristian M, 2006), videre gis en utfyllende informasjon her.

B.1 Formatering av CompactFlash-kortet

Formater minnekortet før installasjon. Sett kortet i en USB stasjon, kjør komandoen *dmesg*, og du vil se hvilken stasjon du har satt inn kortet i. Videre her er det valgt sdc.

Formatere CompactFlash disk fra usb stasjon:
Først må man kjøre komandoen *umount* på stasjonen.

$ *sudo umount /dev/sdc*
$ *sudo mke2fs /dev/sdc*

B.2 BIOS-oppsett

For å boote fra USB pinne, er de viktigste innstillinger her vist:

Advanced BIOS Features
$-> \ First\ Boot\ Device -> \ [USB-FDD]$
$-> \ Boot\ Other\ Devices -> \ [ENABLED]$

Integrated Pheriherals -> On Chip IDE Device
$-> On-Chip\ Serial\ ATA -> \ [DISABLED]$

B.3 Under installasjon

Når du installerer og kommer til partisjoneringen, så velges automatisk partisjonering. Deretter må partisjonen for SWAP slettes. Man kan ikke installere linux på CompactFlash kortet om man prøver å installere med swap. Swap kan ikke brukes på CompactFlash kortet da dette har begrenset med antall ganger det kan leses og skrives til Installeres det med swap, vil man fort ødelegge kortet.

B.4 Autologin

Det har vært ønskelig med automatisk innlogging på roboten, dette for å spare tid når man har koblet til tastatur og skjerm. Automatisk innlogging gjøres ved hjelp av følgende fremgangsmåte:
Man må først installere programmet *rungetty* som ligger i reposotory universe.

$ *sudo apt − get install rungetty*

Deretter må filen */etc/inittab* editeres, og følgende endringer må gjøres. Bytt ut linjen:

1 : 2345 : *respawn* : */sbin/getty* 38400 *tty*1

med:

1 : 2345 : *respawn* : */sbin/rungetty tty*1 *− −autologin eurobot*

der *eurobot* er brukernavnet.

Tillegg C

Installere PCAN-driver på Ubuntu-server

Før noe kan installeres på hovedkortet må man sette opp essensielle program i operativsystemet. Header-filene til kernelen må installeres, og man må lage en symlink fra headerfilene til */usr/src/linux*. Symlinken må lages da pcan makefilen leter etter hedder-filene under */usr/src/linux*.

$*sudo\ apt-get\ install\ linux-headers-*\$(*uname\ -r*)
$*cd\ /usr/src*
$*sudo\ ln\ -s\ /usr/src/linux-headers-*\$(*uname\ -r*)\ *linux*

$*sudo\ apt-get\ install\ gcc*
$*sudo\ apt-get\ install\ make*
$*sudo\ apt-get\ install\ build-essential*

Nå er systemet konfigurert for å installere driveren. Last ned nyeste driver fra:

http://www.peak-system.com/linux/

Legg driveren i en mappe på brukerområdet.
Du må koble til den pcan adapteren du skal bruke til maskinen den skal installeres på.

Pakk ut og installer driveren driveren.
$*tar\ -xzf\ peak-linux-driver-*5.6*.tar.gz*
$*cd\ peak-linux-driver-*5.6
$*sudo\ make\ clean*
$*sudo\ make*
$*sudo\ make\ install*

Trekk nå ut og sett PCAN enheten på USB porten inn igjen.

Nå er driveren installert. Du kan se hvilket filnavn den er gitt under $/dev$.
$\$ls\ -l\ /dev/pc*$
$cr--r--r--\ 1\ root\ root\ 180,\ 32\ 2007-01-25\ 10:58\ /dev/pcan0$

For å kunne skrive og lese fra enheten må rettigheter settes:
$\$sudo\ chmod\ 666\ /dev/pcan0$

Nå kan driveren testes. Her sendes standard CAN-melding med identifikator 0x120, meldingslengde 2 byte med meldingsdata 0x55 og 0x44.
$\$sudo\ echo\ 'm\ s\ 0x120\ 0x02\ 0x55\ 0x44'\ >\ /dev/pcan0$

Feilmeldinger
Man kan få diverse feilmeldinger om at enkelte interfaces ikke er støttet. Man kan disable disse interfaces når man kompilerer med make:

$\$sudo\ make\ PCC = NO\_PCC\_SUPPORT$

Her disables PC-CARD support.

Tillegg D

CAN-BUSS-Adresser

Her er gitt en oversikt over hvilke CAN-BUSS-adresser som er benyttet på de forskjellige modulene av systemet. CAN-BUSS-adressene er deffinert i PROXY-/GATEWAY-programmet i filen *proxyconfig.h*

```
CAN_HIGHEST_ID 0x7FF


CAN_GAME_STOP 0x05
CAN_INIT_GAME_TIME 0x10
CAN_RESET_GAME 0x11

// Colition detection 0x80 - 0x8F
CANID_COLLISION_DETECTED 0x80
CANID_COLLISION_SLAVE_MASTER_COMMANDS 0x81
CANID_RANGE_MEASUREMENTS 0x84
CANID_REQUEST_MEASUREMENTS 0x85
CANID_FLASH_SENSOR_BOARD_LED 0x86
CANID_ENABLE_SENSOR_LEDS 0x87
CANID_SET_COLLISION_THRESHOLDS 0x88
CANID_SET_INTERRUPT_REFRESH_INTERVAL 0x89
CANID_ENABLE_SENSOR_LIGHTS 0x87

// MOTOR
CAN_ADDR_MOTOR_MAIN 0x110

// Motor RIGHT 0x115 - 0x120
CAN_ADDR_MOTOR_RIGHT_SETDIR 0x116
CAN_ADDR_MOTOR_RIGHT_SETSPEED 0x117
CAN_ADDR_MOTOR_RIGHT_READ_QUAD 0x118
CAN_ADDR_MOTOR_RIGHT_RETURN_QUAD 0x71

CAN_ADDR_MOTOR_DRIVER_INIT 0x11D
CAN_ADDR_MOTOR_EMERGENCY_STOP 0x11E
CAN_ADDR_MOTOR_INIT_QUAD 0x11F
```

```
// Motor LEFT 0x111 - 0x114
CAN_ADDR_MOTOR_LEFT_SETDIR 0x111
CAN_ADDR_MOTOR_LEFT_SETSPEED 0x112
CAN_ADDR_MOTOR_LEFT_READ_QUAD 0x113
CAN_ADDR_MOTOR_LEFT_RETURN_QUAD 0x70

// IO CARD Quadrature 0x130 - 0x13F
CAN_CMD_SET_TEAM_COLOR_RED 0x16
CAN_CMD_SET_TEAM_COLOR_BLUE 0x15

CAN_ADDR_READ_DUAL_QUAD 0x133
CAN_ADDR_INIT_DUAL_QUAD 0x13F
CAN_ADDR_RETURN_DUAL_QUAD 0x72

CAN_SET_LED_TEAM_COLOR_RED 0x135
CAN_SET_LED_TEAM_COLOR_BLUE 0x134
CAN_ADDR_RELEASE_BATTERY 0x13A
CAN_ADDR_BATTERY_RELEASED 0x13B


// Bottle Box Sort 0x180 - 0x18F
CAN_ADDR_PICK_UP_BOTTLE_BOX 0x183
CAN_ADDR_RELEASE_BOTTLE_BOX 0x184
CAN_ADDR_BOTTLE_BOX_RELEASED 0x185
CAN_ADDR_BOTTLE_BOX_TYPE 0x186
CAN_ADDR_BOTTLE_BOX_STOP_GAME 0x187
CAN_ADDR_PICK_UP_BOTTLE_SUCCESS 0x53
CAN_ADDR_PICK_UP_BOX_SUCCESS 0x189
CAN_ADDR_BOTTLE_BOX_PICK_UP_FAILURE 0x18A
CAN_ADDR_BOTTLE_BOX_START_GAME 0x18B
```

Tillegg E

Møtereferat fra møte med Keith Downing

Tilstede: Keith Downing, Lars Vråle. og Gunnar Kjemphol.
Dato: 22. januar 2007.
Møtetid: 13.00.
Møtested: Rom 308 IT-Vest.

Sammendrag
Etter å ha kontaktet leder av «AI»-gruppen til NTNU, Agnar Aamodt, fikk undertegnede og Lars Vråle et møte med Keith Downing. Downing jobber med kunstig intelligens, kunstig liv og computer nevrovitenskap (Keith Downing, 2007).
Downing fikk en rask innføring i Eurobot, samt en del spørsmål om hvordan man skal gå frem for et best mulig planleggingssystem for roboter, da særlig rettet mot Eurobot konkurransen.
Lars Vråle og undertegnede ble deretter raskt underrettet om biologisk og kunstig liv, blant annet hvordan mauren navigerer og finner tilbake dit den før har vært. Dette gjøres ved at den sammenligner det den ser med bilder i hukommelsen.

Etter å ha diskutert en stund ble undertegnede anbefalt å studere arbeide av Rodney Brooks, og Pattie Maes ved MIT. Det er Brooks arbeid med «Subsumption Architecture», og Maes arbeid med «Action selection mechanism» som er interessante.

Mot slutten av møtet ønsket Downing undertegnede og Vråle lykke til med arbeidet. Downing ønsket også å se roboten sammen med noen studenter når roboten nærmer seg ferdigstillelse.

Gunnar Kjemphol

Tillegg F

Produksjon av printkort

Fremgangsmåte for produksjon av printkort ved hjelp av utstyret til ITK.

Før man går igang med selve printkort fremstillingen må man printe ut transparent med kretskort utlegget. Transparenten kan godt printes ut på laserskriver, selv om dette ikke gir like tett farge som en blekkskriver. Man må også gå til anskaffelse av printkort med fotobelegg. Skal man produsere tosidig kort printer man to transparenter, en til hver side og tape disse sammen.

Omtrent 20 minutter før man ønsker å starte etsing og fremkalling av prinkortet må, etsebadet og fremkaller maskinen skrues på, da disse trenger oppvarming. Temperatur innstillingen på fremkallermaskinen, kan justeres på et hjul i front av maskinen. Dette hjulet har en sort prikk som skal stå i posisjon klokken to, sett fra en klokke. Fremkaller vesken blir varm iløpet av noen minutter, og etsebadet bruker omtrent 20 minutter på oppvarming. Når de to badene er varme, kan man starte prinkort fremstilling.

- Skru av alt annet enn det gule lyset på etserommet.
- Fjerner fotobeskyttelsesfilmen på printkortet, og legg kortet mellom de to transparentene. Det er viktig å passe på at siden med toner på transparenten kommer inn mot prinkortet.
- Legg kortet i belysnigsmaskinen, og fest låseklemmene.
- Belys kortet i 190 sekunder.

- Legg deretter kortet i fremkallerbadet i 45 til 60 sekunder, man skal klart se printbanene før kortet tas ut.
- Skyll kortet godt i vann etter fremkallingen.

- Sett kortet fast i etsemaskinen, og start denne.
- Etsing tar omtrent tre minutter, men man må selv kontrollere og være sikker på at alt som skal etses bort er borte. Pass på å ikke etse lenger enn dette, da de tynne banene på kortet vil begynne å forsvinne.
- Etter fremkalling skylles kortet i vann.

- Belys kortet 190 sekunder, og fremmkall deretter igjen. Dette er for å få bort det resterende av fotobelegget.
- Kortet skal deretter kaldfortinnes. Dette gjøres på rommet innenfor etserommet. Her står en hvit 10 liter kanne med surtinn. Legg kortet i en hvit bøtte som står der, og hell surtinn over til kortet akkurat dekkes.
- Kortet skal ligge i surtinn i 3 til 5 minutter, til alt kobberet har fått sølvfarge.

- Nå er kortet ferdig produsert og klart for boring. Bruk 0.9 mm bor for standard komponenter, og viapinner.
- Viapinner settes i gjennomgangene på printkortet, med presseverktøyet som står vedsiden av boremaskinen.

- Kortet er da ferdig produsert og klart for lodding.

Tillegg G

Proforma Faktura

Institutt for Teknisk Kybernetikk
v/Eurobot
O.S. Bragstads Plass 2D
7034 Trondheim
NORWAY

MaxBotix Inc.
1821 Graydon Ave
Brainerd, MN 56401
United States

Proforma Invoice

1. LV-MaxSonar-EZ1 5-pack $124,75 USD

NO CHARGE.
VALUE FOR CUSTOM ONLY:
RETURNED FOR REPAIR .

RMA Number: RMA05607

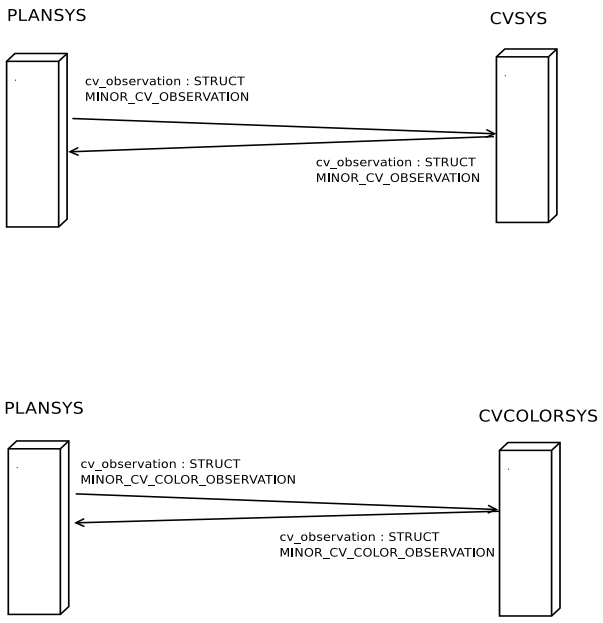
09.03.2007

Gunnar Kjemphol

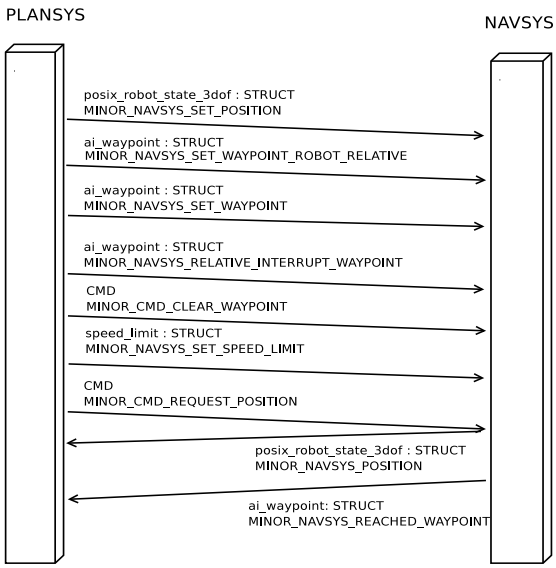
Figur G.1: Proformafaktura for utførsel av varer til reperasjon

Tillegg H

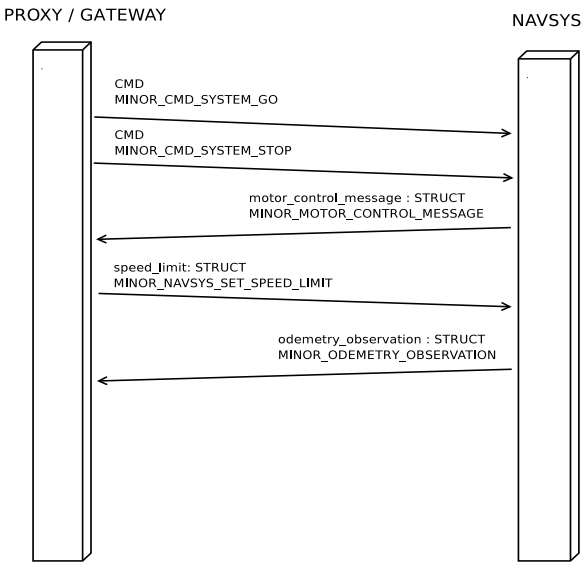
Grensesnitt



Figur H.1: Grensesnitt mellom planleggingssystem og bildebehandlingssystemene



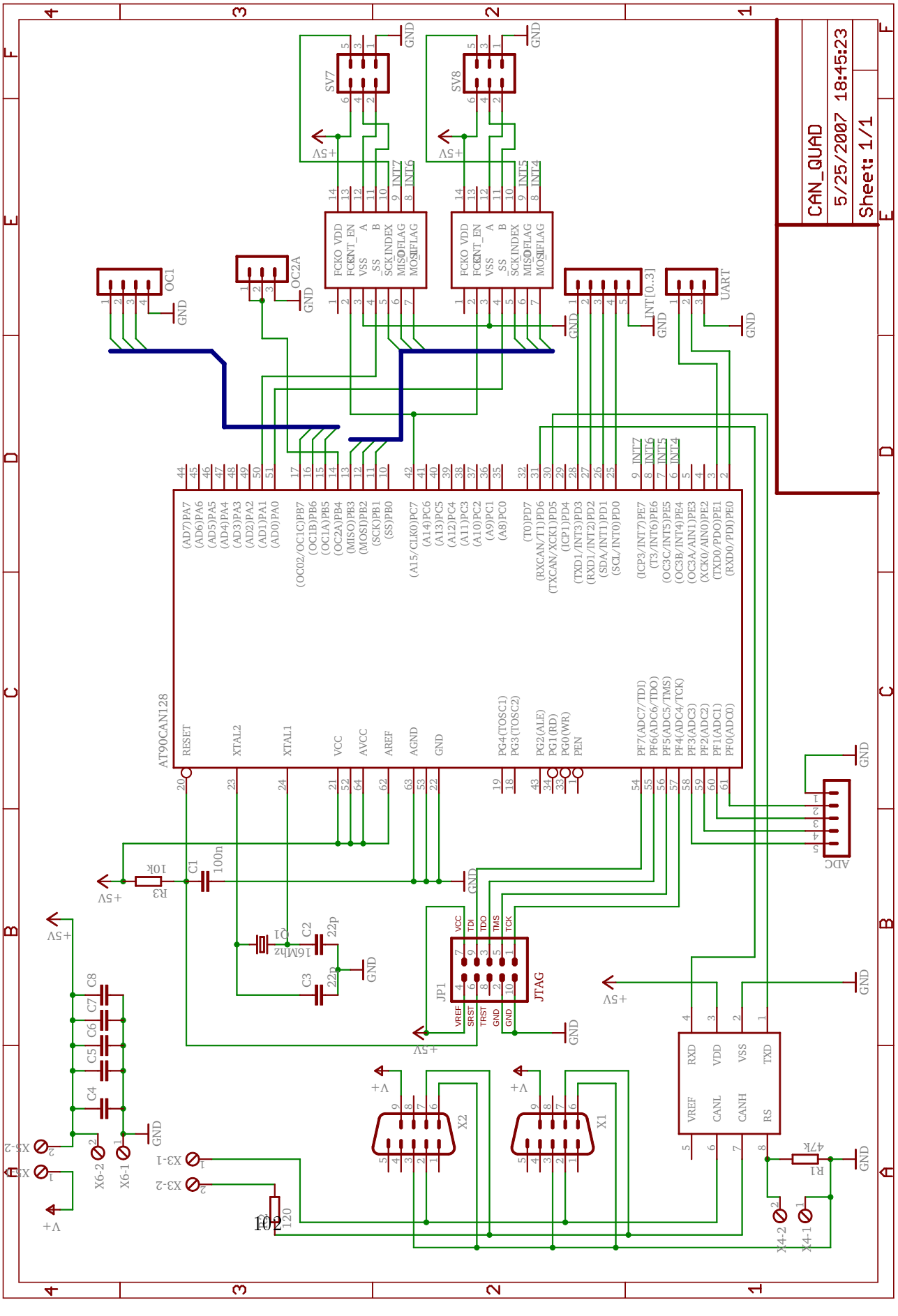
Figur H.2: Grensesnitt mellom planleggingssystem og navigasjonssystemet



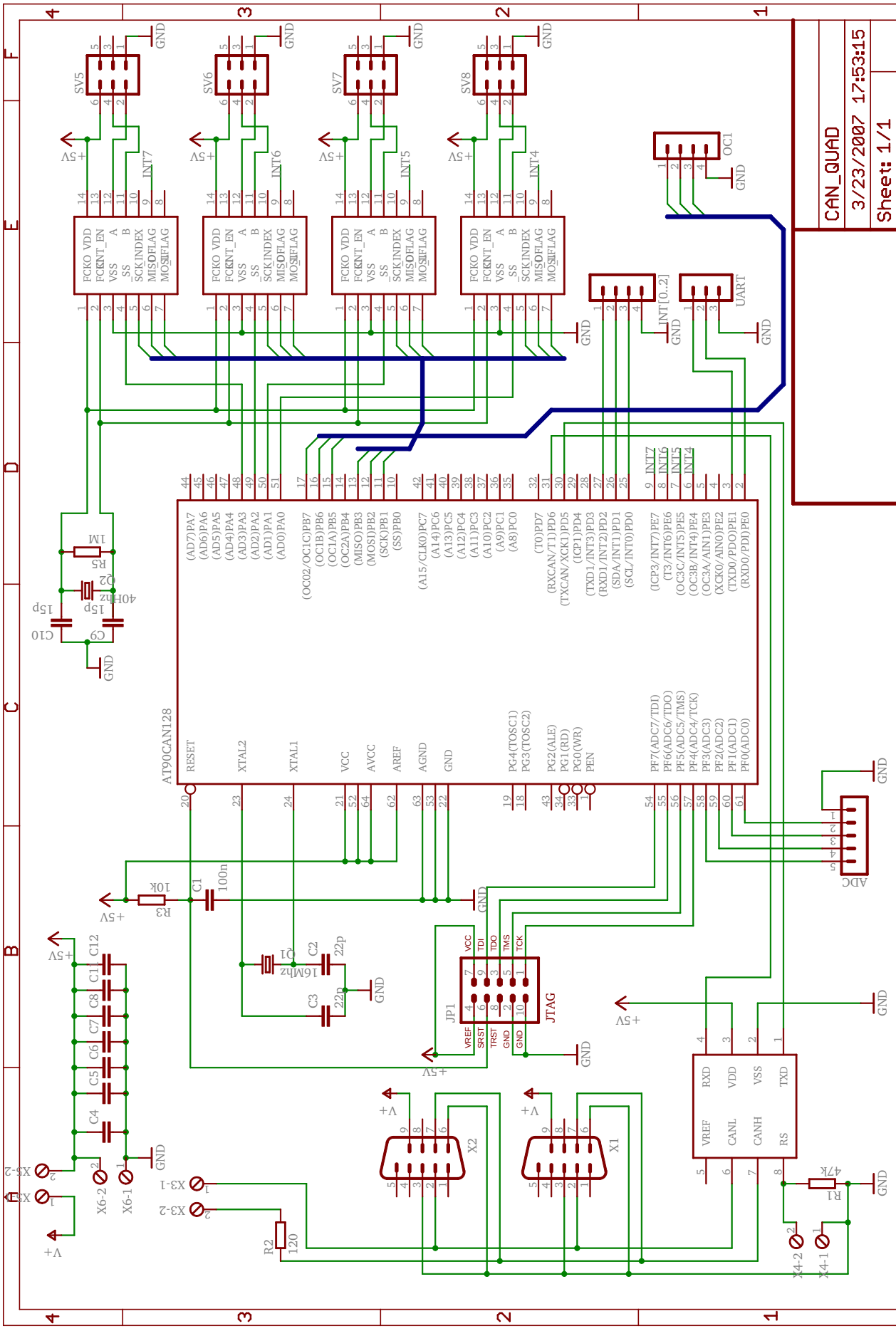
Figur H.3: Grensesnitt mellom proxy-/gateway-systemet og navigasjonssystemet

Tillegg I

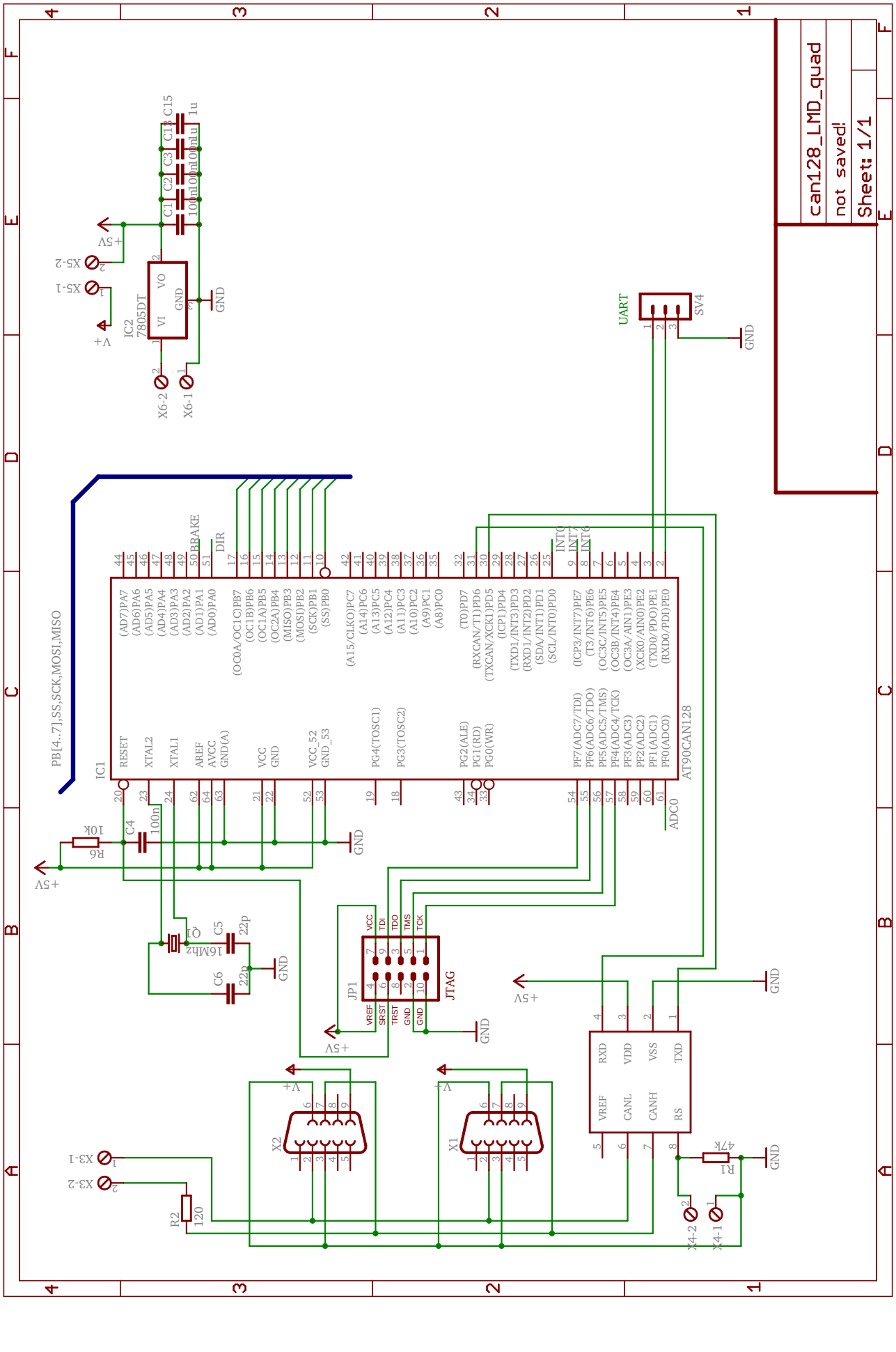
Kretstegninger



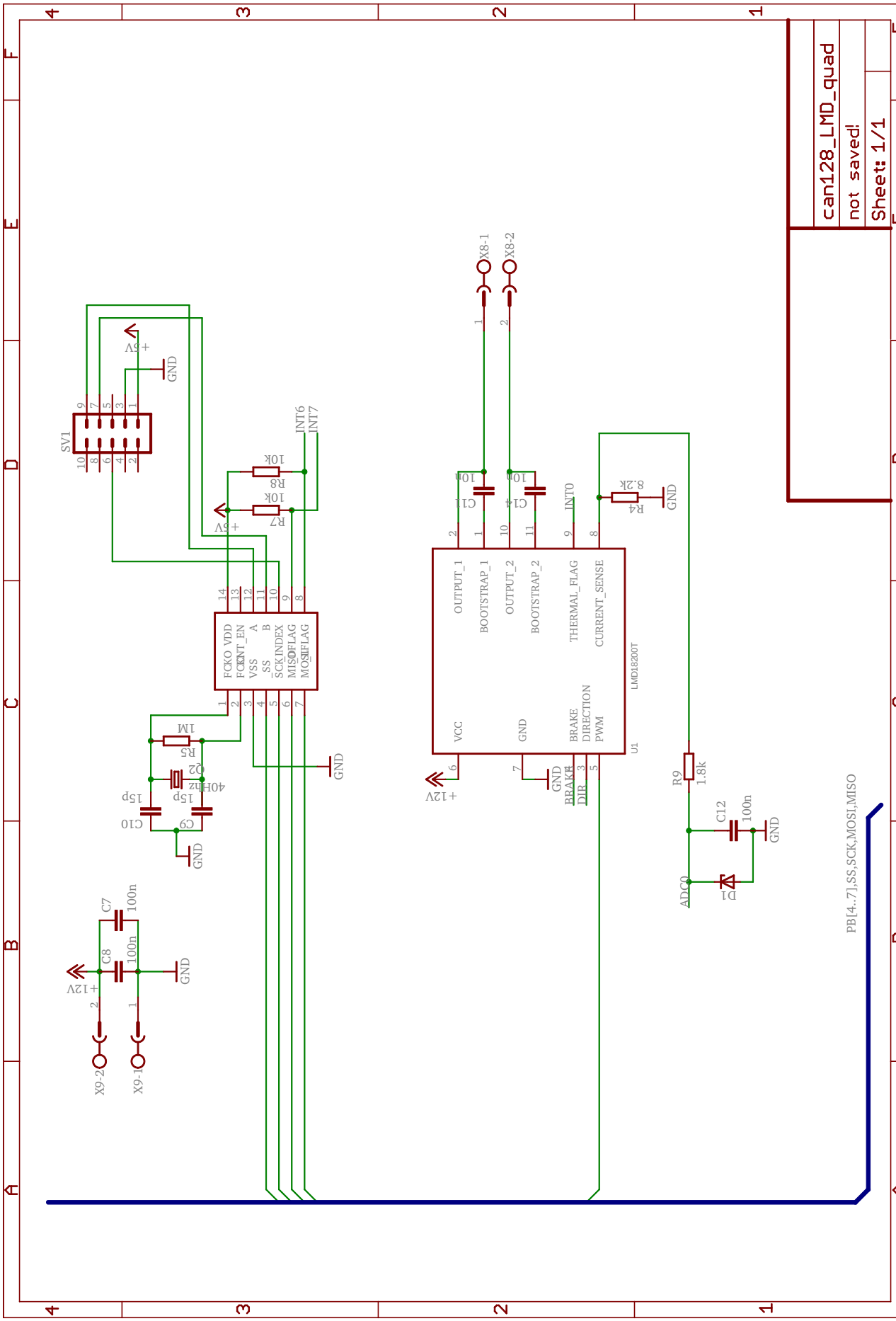
Figur I.1: IO kort med to kvadraturtellere



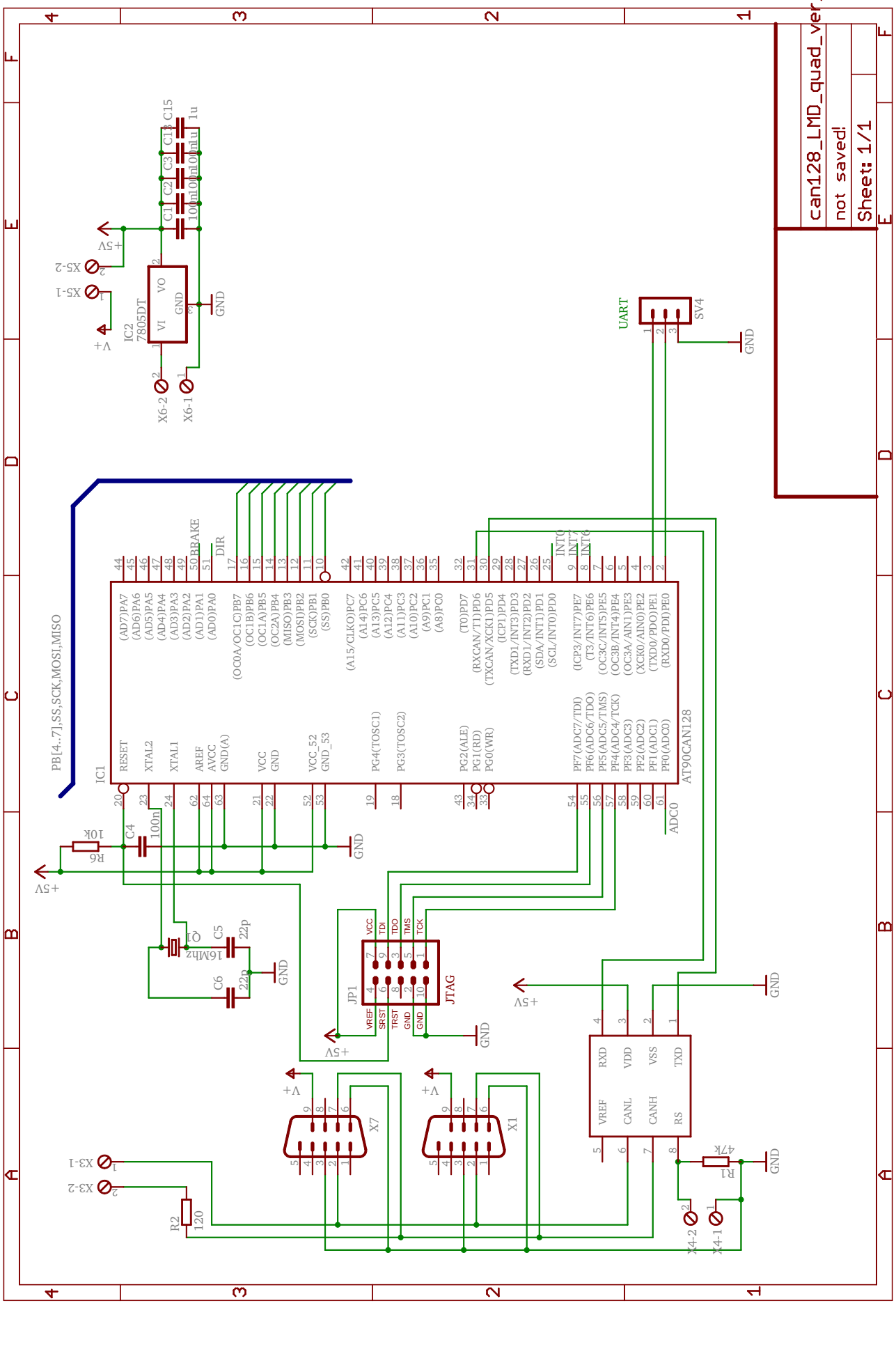
Figur I.2: IO kort med fire kvadraturtellere



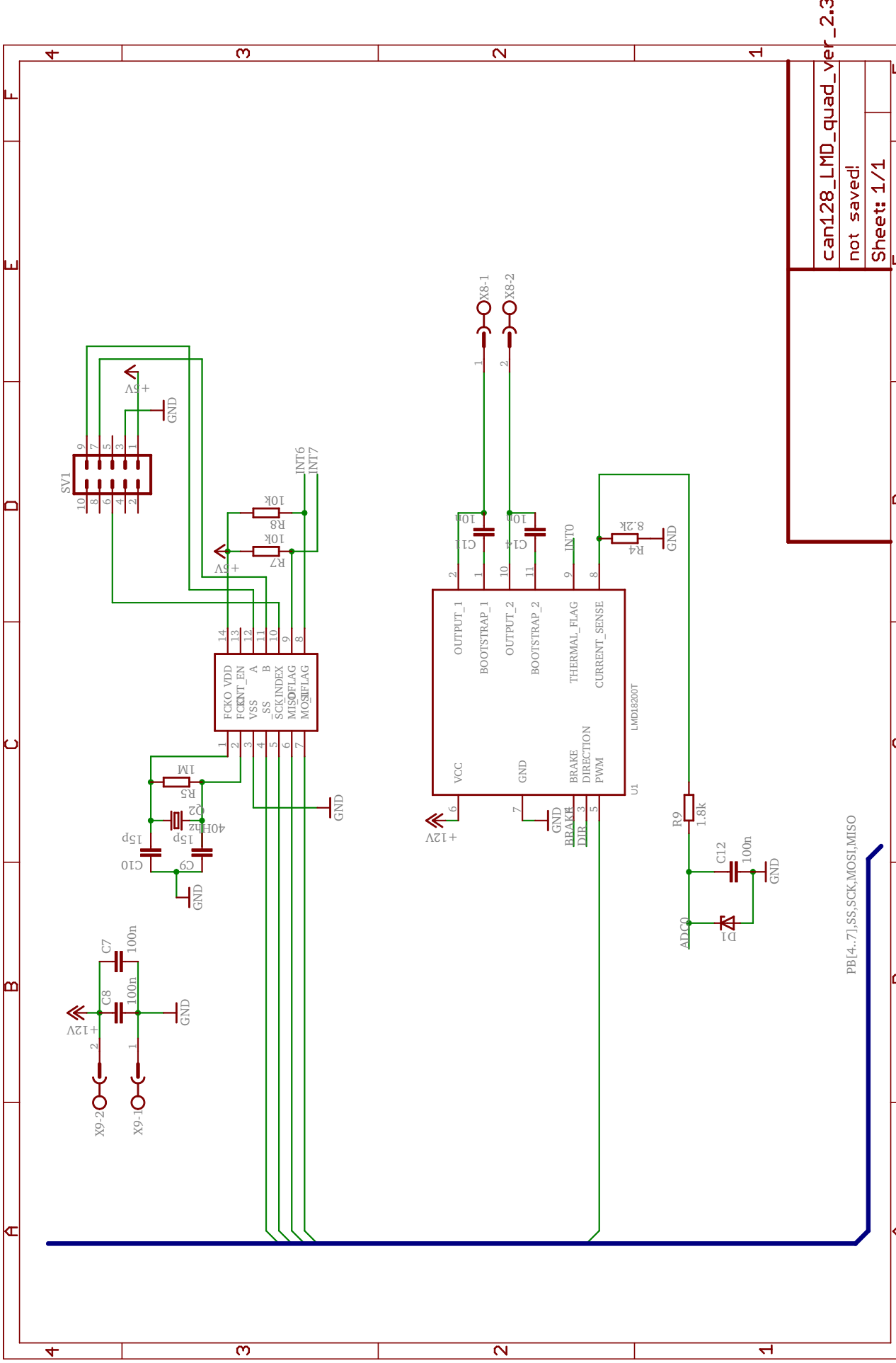
Figur I.3: Motordriver version 2.2 Del 1



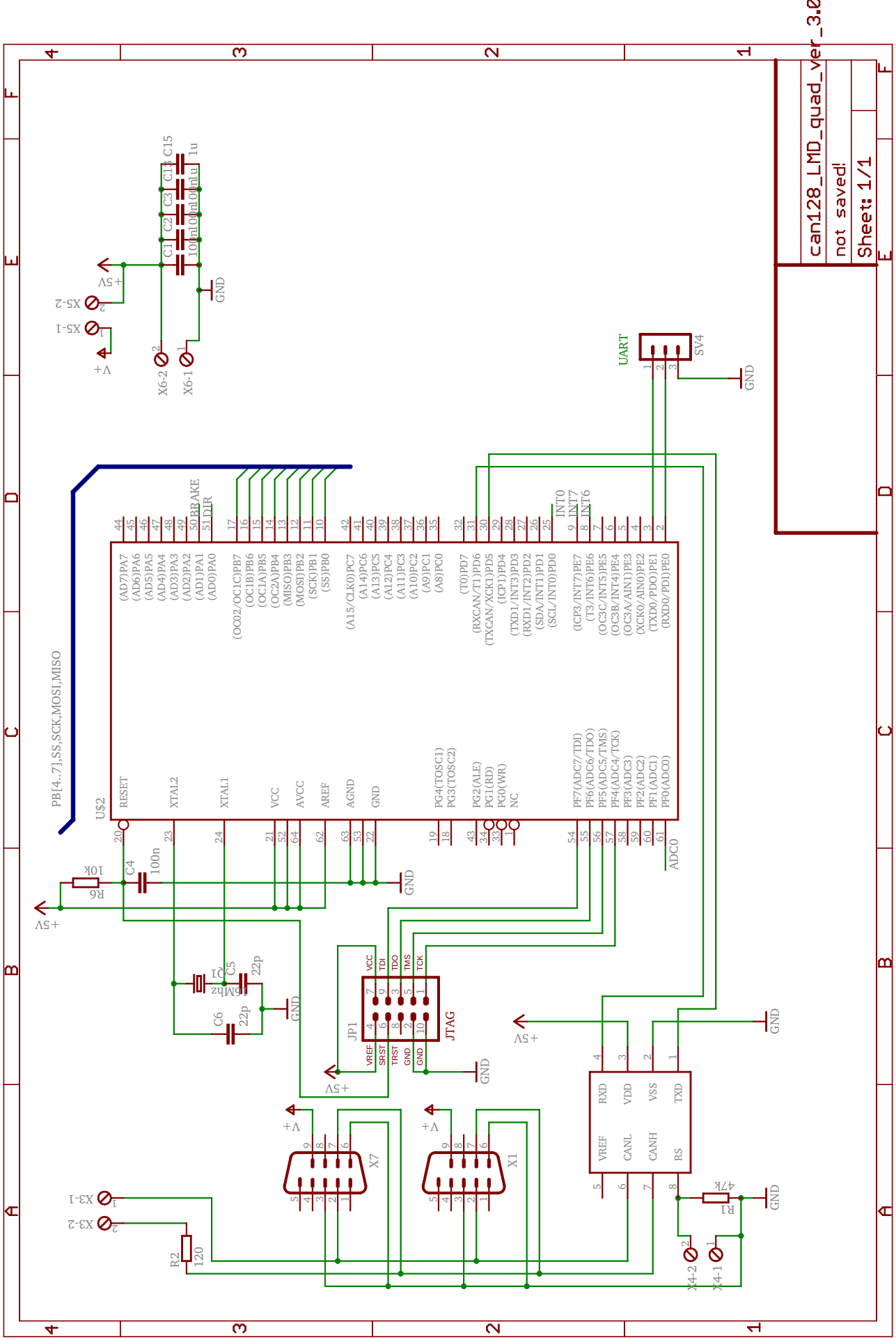
Figur I.4: Motordriver version 2.2 Del 2



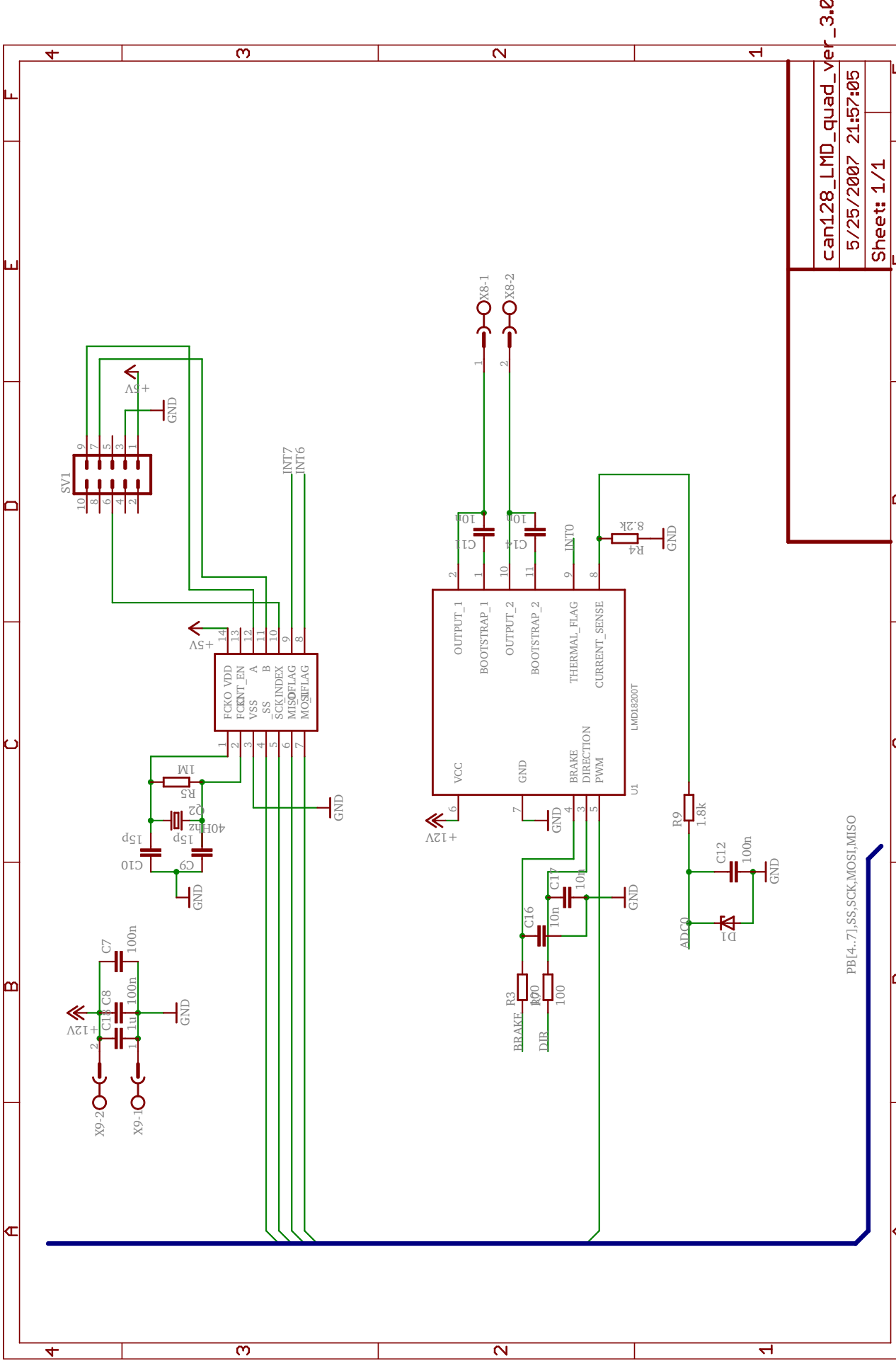
Figur I.5: Motordriver version 2.3 Del 1



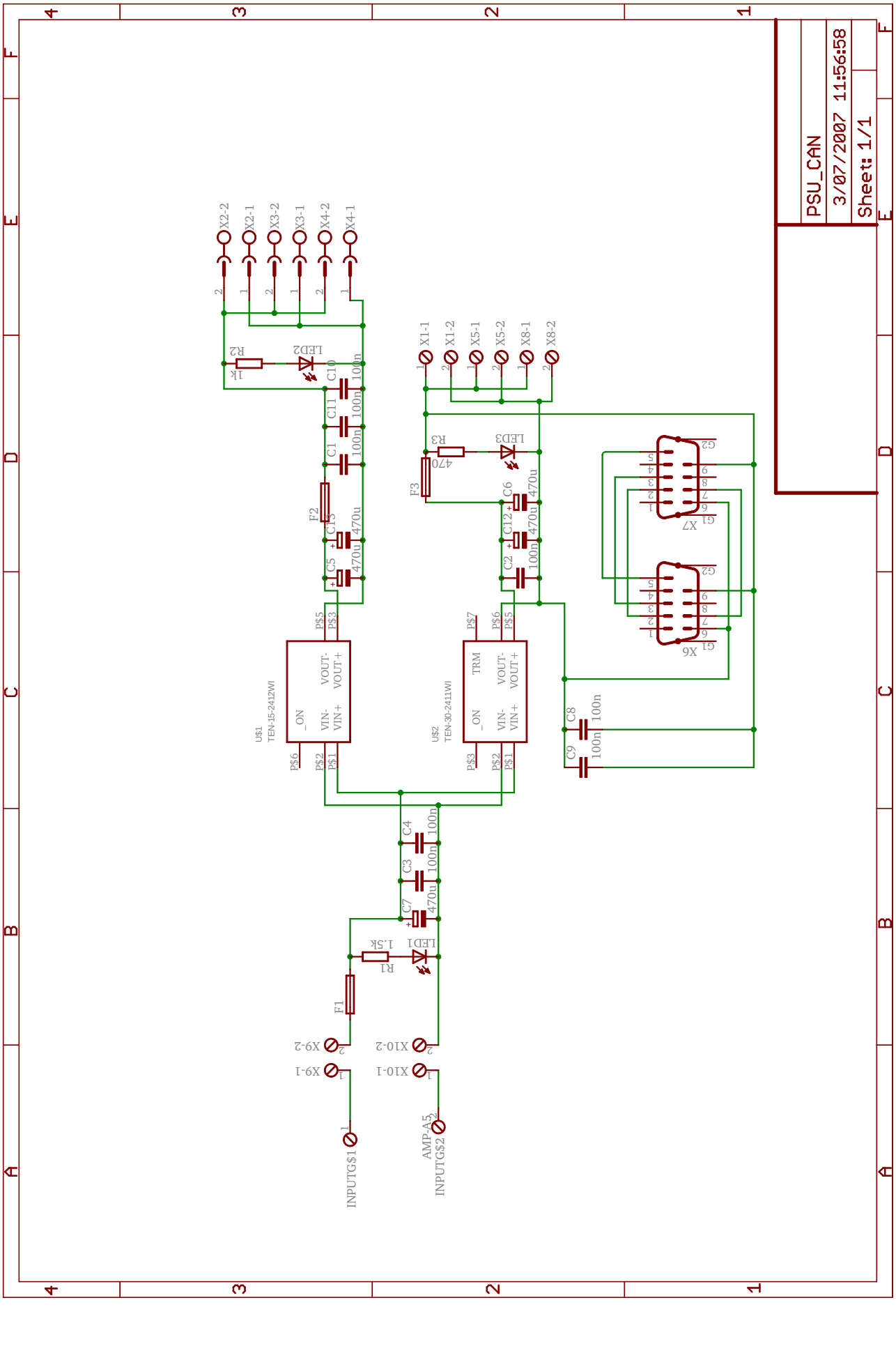
Figur I.6: Motordriver version 2.3 Del 2



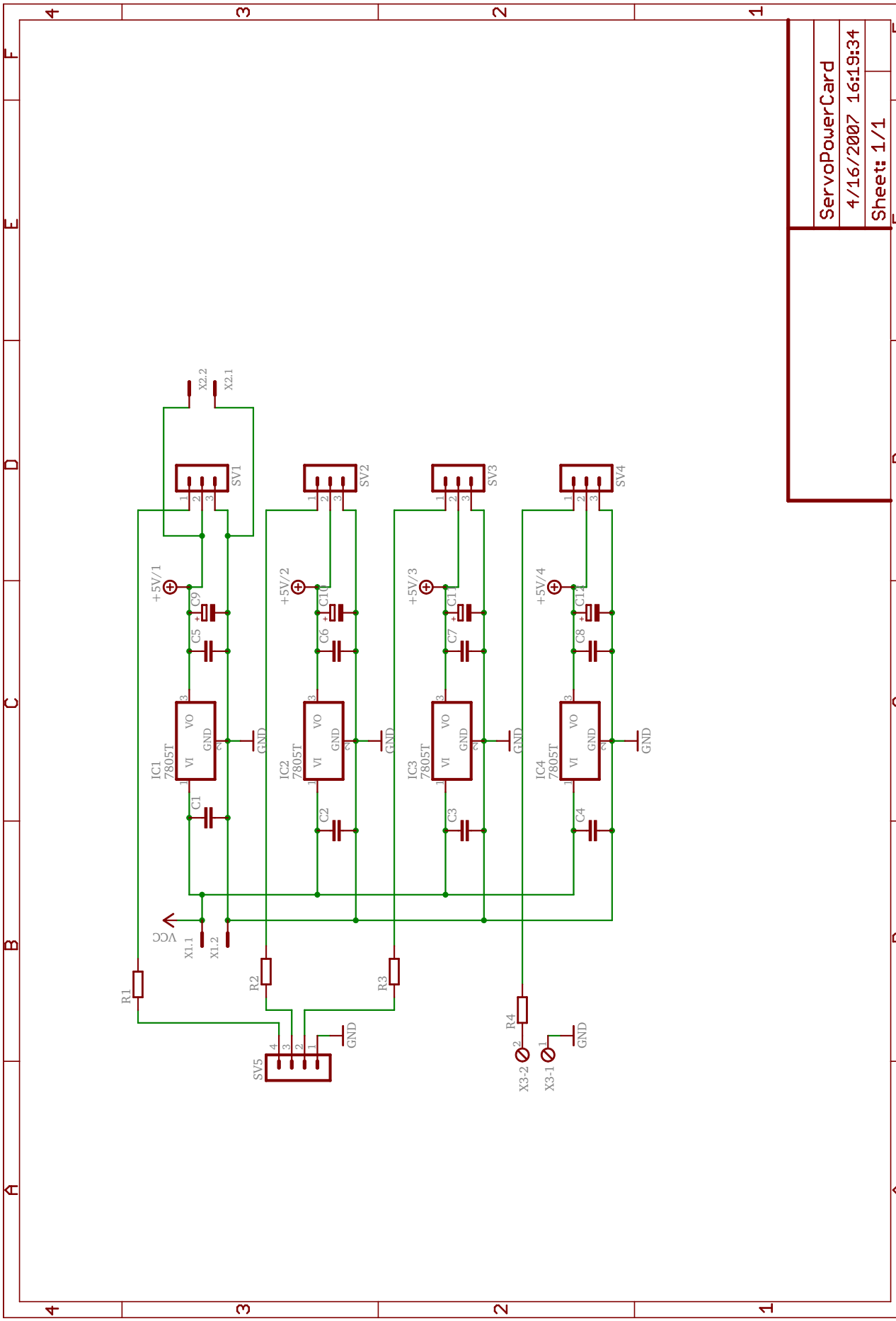
Figur I.7: Motordriver version 3.0 Del 1



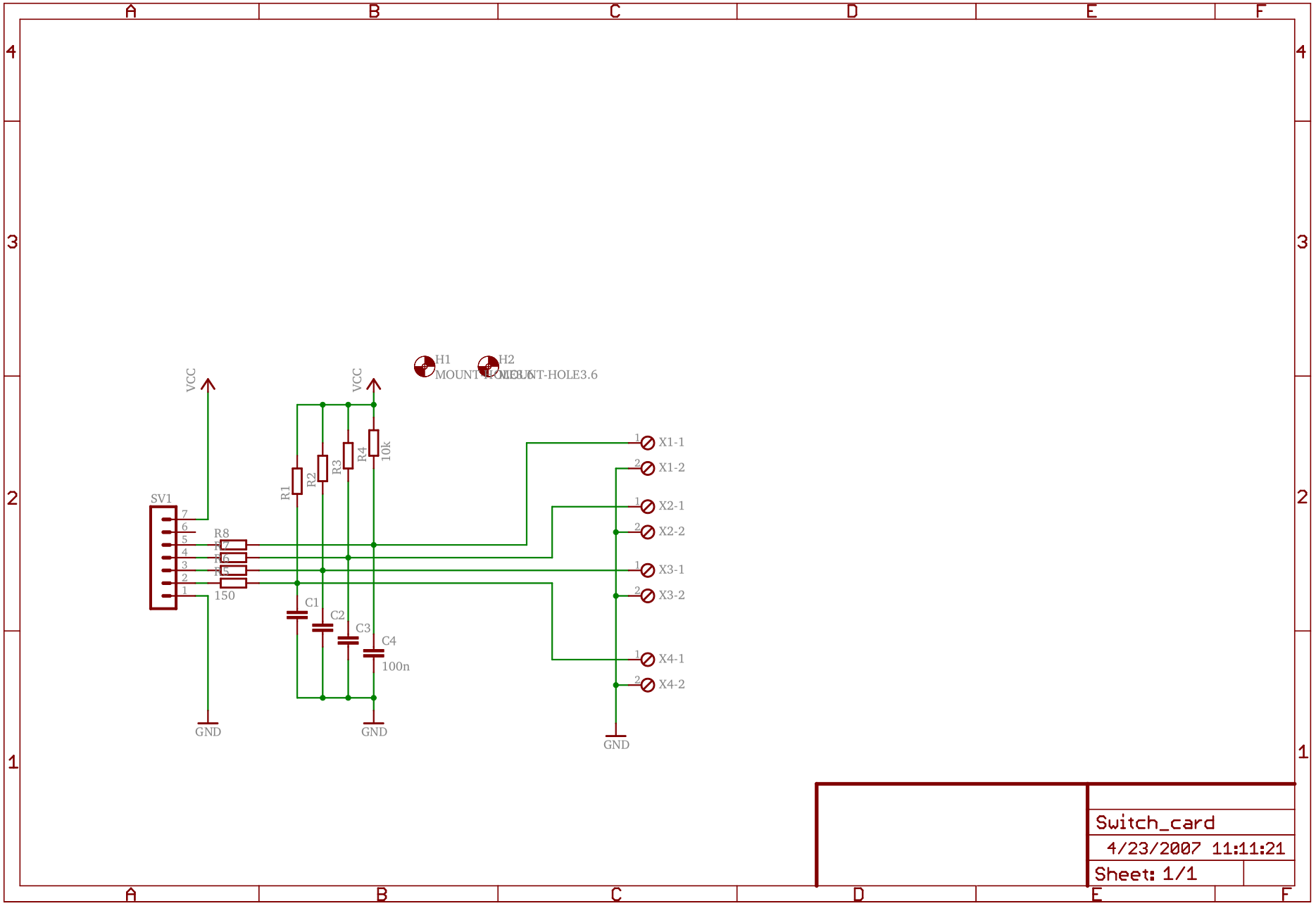
Figur I.8: Motordriver version 3.0 Del 2



Figur I.9: Switch-mode spenningsforsyning



Figur I.10: Servo spenningsforsyning



Figur I.11: Trykknapp kort

5/25/2007 21:14:39 /home/gunnar/eurobotsvn/eurobot/Kretskort(CAD)/switch_card/Switch_card.sch (Sheet: 1/1)

Tillegg J

CD-ROM

- Robot->modules - Kildekode for robot.
  - plansys - Planleggingsystemet
  - proxy - Proxy-/Gateway-systemet
- Robot - Felles include filer
  - include
- Kode - Mikrokontroller kode + div. kode.
  - IO_card_with_2quad
  - Motordriver_ver3.0
  - Heyes-Jones_AStar_demo
- EagleCAD - Kretstegninger.
- EagleLibrary - Bibliotek for Eagle CAD
- Regler - Eurobot reglement